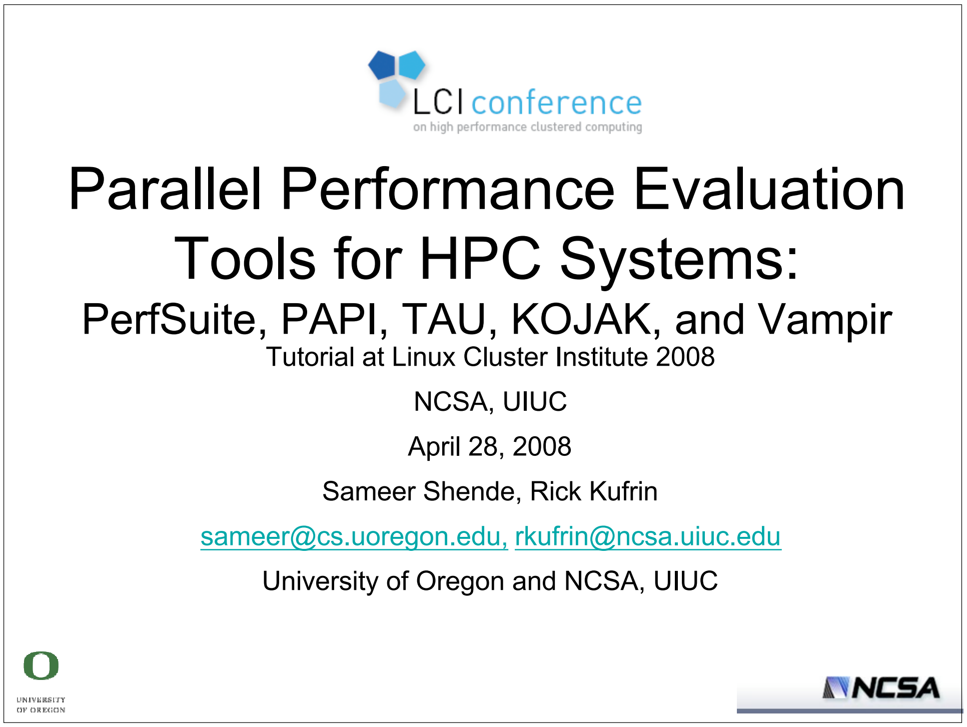


LCI conference
on high performance clustered computing

Parallel Performance Evaluation Tools for HPC Systems:

PerfSuite, PAPI, TAU, KOJAK, and Vampir

Tutorial at Linux Cluster Institute 2008

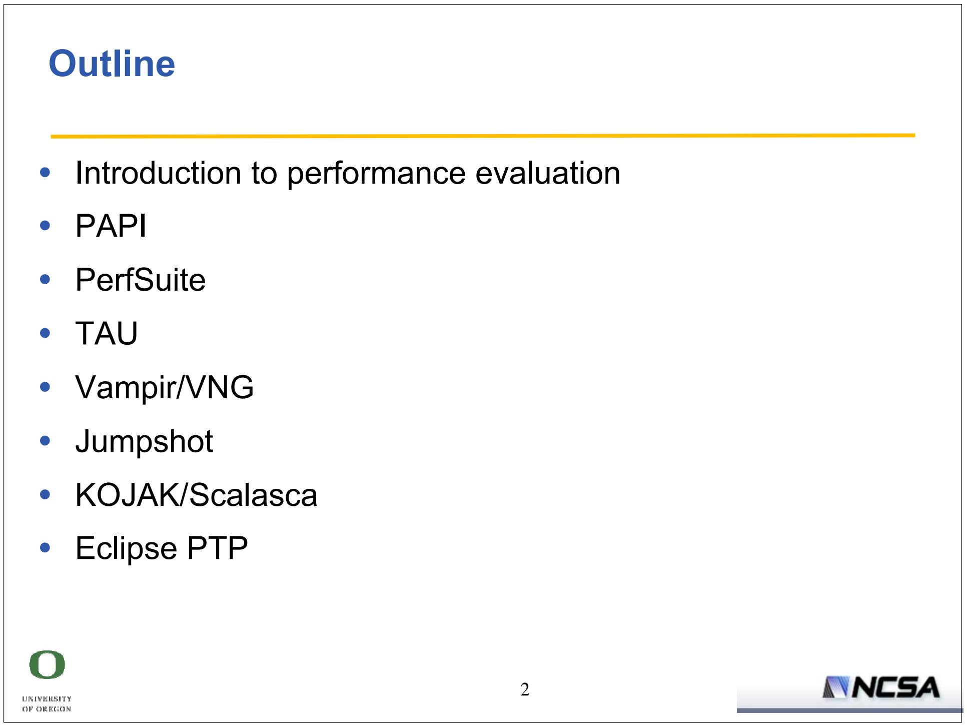
NCSA, UIUC

April 28, 2008

Sameer Shende, Rick Kufrin

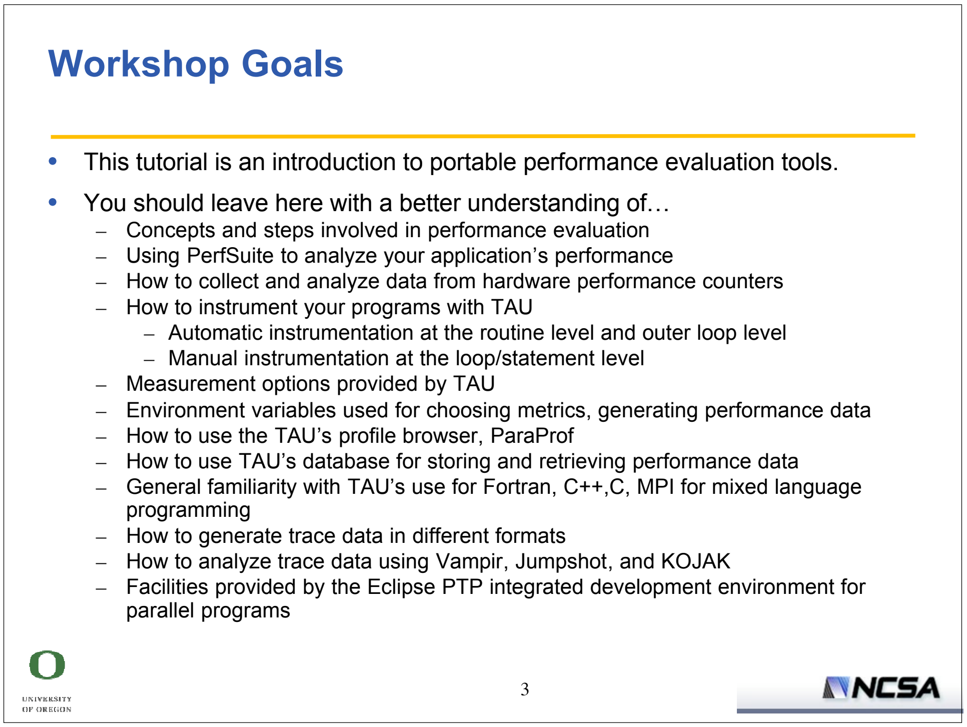
sameer@cs.uoregon.edu, rkufrin@ncsa.uiuc.edu

University of Oregon and NCSA, UIUC



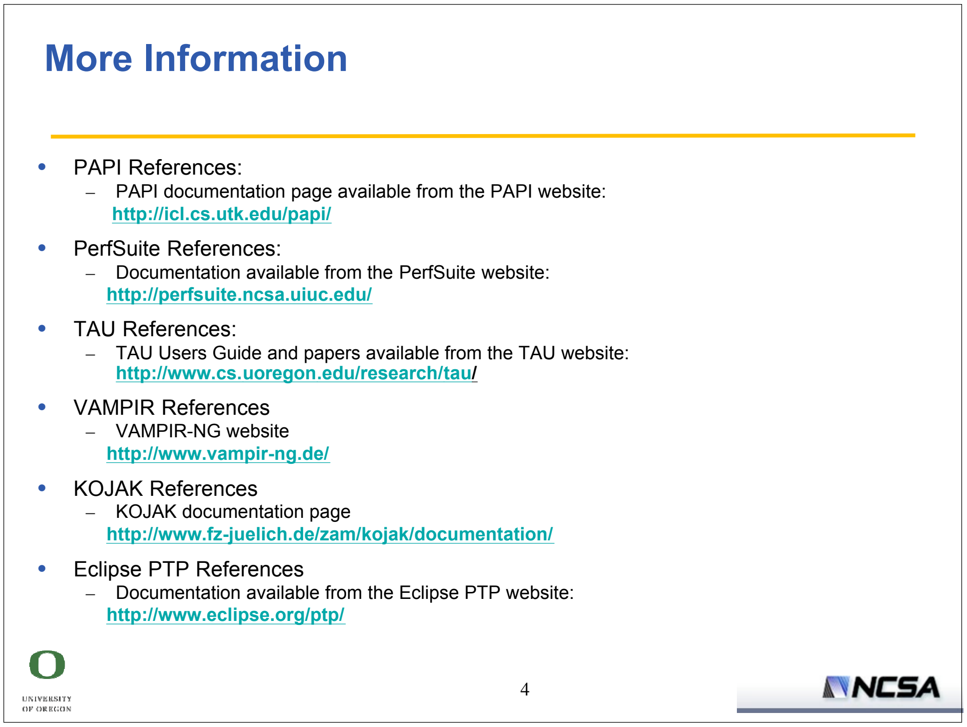
Outline

- Introduction to performance evaluation
- PAPI
- PerfSuite
- TAU
- Vampir/VNG
- Jumpshot
- KOJAK/Scalasca
- Eclipse PTP



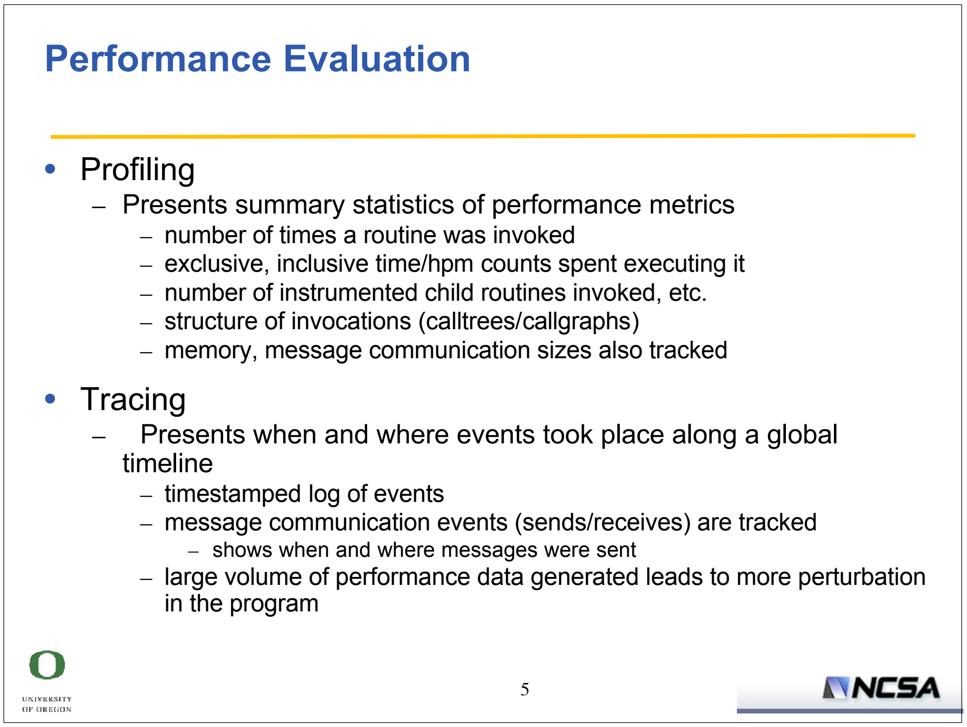
Workshop Goals

- This tutorial is an introduction to portable performance evaluation tools.
- You should leave here with a better understanding of…
  - Concepts and steps involved in performance evaluation
  - Using PerfSuite to analyze your application's performance
  - How to collect and analyze data from hardware performance counters
  - How to instrument your programs with TAU
    - Automatic instrumentation at the routine level and outer loop level
    - Manual instrumentation at the loop/statement level
  - Measurement options provided by TAU
  - Environment variables used for choosing metrics, generating performance data
  - How to use the TAU's profile browser, ParaProf
  - How to use TAU's database for storing and retrieving performance data
  - General familiarity with TAU's use for Fortran, C++,C, MPI for mixed language programming
  - How to generate trace data in different formats
  - How to analyze trace data using Vampir, Jumpshot, and KOJAK
  - Facilities provided by the Eclipse PTP integrated development environment for parallel programs



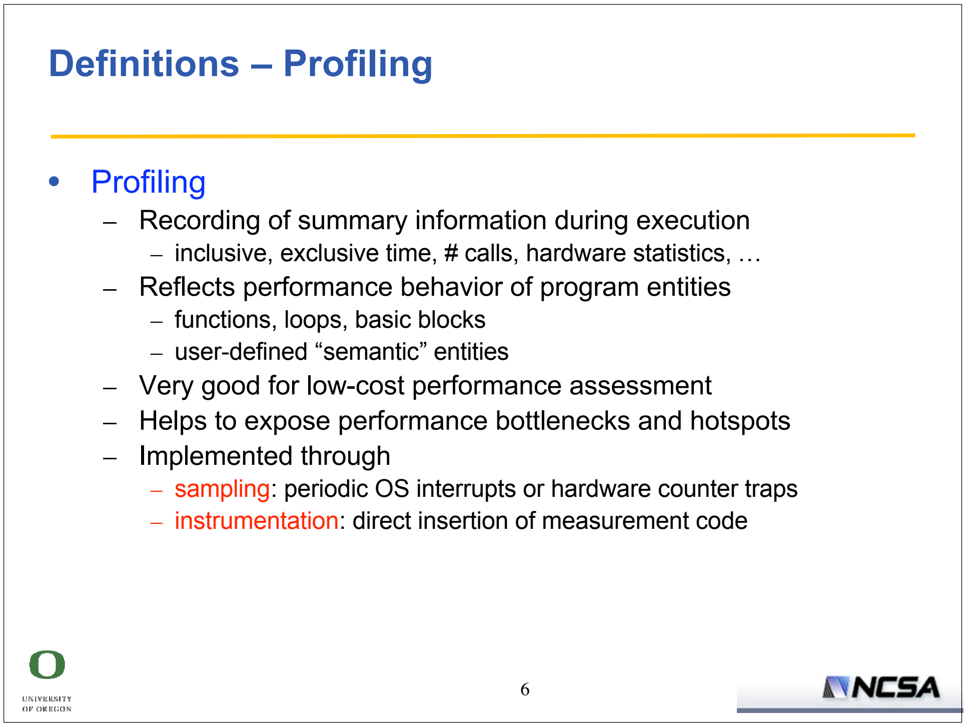
More Information

- PAPI References:
  - PAPI documentation page available from the PAPI website: http://icl.cs.utk.edu/papi/
- PerfSuite References:
  - Documentation available from the PerfSuite website: http://perfsuite.ncsa.uiuc.edu/
- TAU References:
  - TAU Users Guide and papers available from the TAU website: http://www.cs.uoregon.edu/research/tau/
- VAMPIR References
  - VAMPIR-NG website http://www.vampir-ng.de/
- KOJAK References
  - KOJAK documentation page http://www.fz-juelich.de/zam/kojak/documentation/
- Eclipse PTP References
  - Documentation available from the Eclipse PTP website: http://www.eclipse.org/ptp/



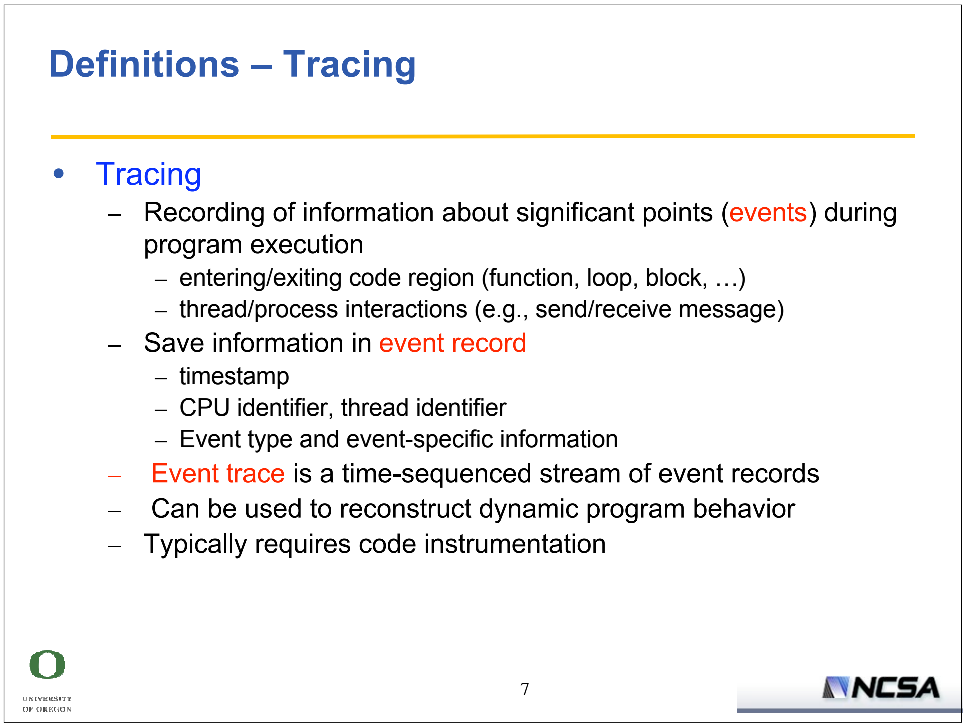
Performance Evaluation

- Profiling
  - Presents summary statistics of performance metrics
    - number of times a routine was invoked
    - exclusive, inclusive time/hpm counts spent executing it
    - number of instrumented child routines invoked, etc.
    - structure of invocations (calltrees/callgraphs)
    - memory, message communication sizes also tracked
- Tracing
  - Presents when and where events took place along a global timeline
    - timestamped log of events
    - message communication events (sends/receives) are tracked
      - shows when and where messages were sent
    - large volume of performance data generated leads to more perturbation in the program



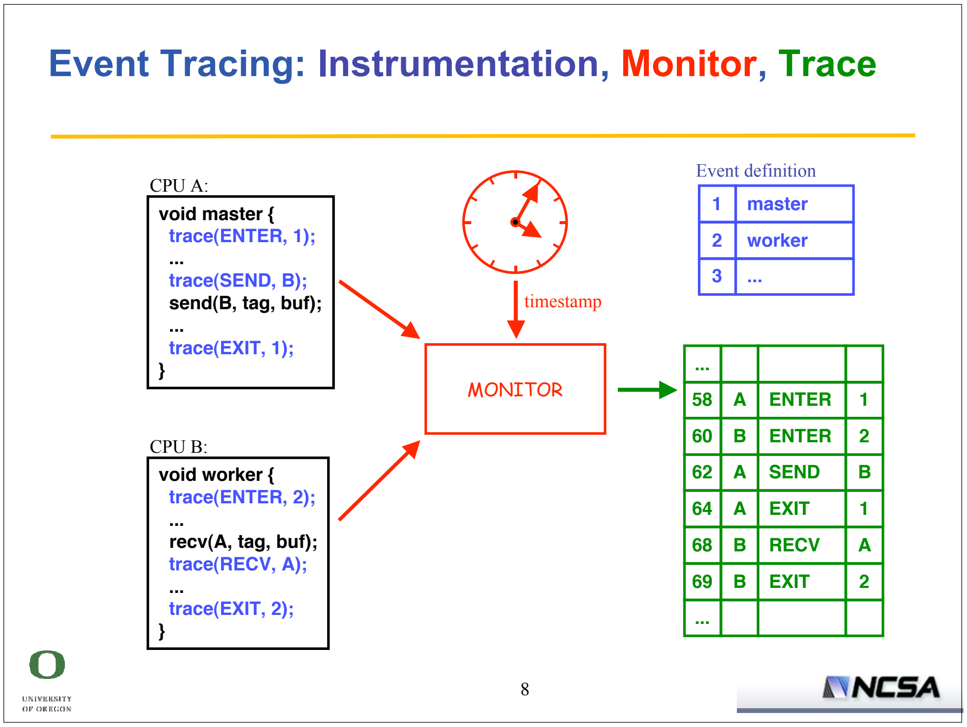
Definitions – Profiling

- Profiling
  - Recording of summary information during execution
    - inclusive, exclusive time, # calls, hardware statistics, …
  - Reflects performance behavior of program entities
    - functions, loops, basic blocks
    - user-defined “semantic” entities
  - Very good for low-cost performance assessment
  - Helps to expose performance bottlenecks and hotspots
  - Implemented through
    - sampling: periodic OS interrupts or hardware counter traps
    - instrumentation: direct insertion of measurement code



Definitions – Tracing

- Tracing
  - Recording of information about significant points (events) during program execution
    - entering/exiting code region (function, loop, block, …)
    - thread/process interactions (e.g., send/receive message)
  - Save information in event record
    - timestamp
    - CPU identifier, thread identifier
    - Event type and event-specific information
  - Event trace is a time-sequenced stream of event records
  - Can be used to reconstruct dynamic program behavior
  - Typically requires code instrumentation



Event Tracing: Instrumentation, Monitor, Trace

CPU A:

```
void master {
  trace(ENTER, 1);
  ...
  trace(SEND, B);
  send(B, tag, buf);
  ...
  trace(EXIT, 1);
}
```

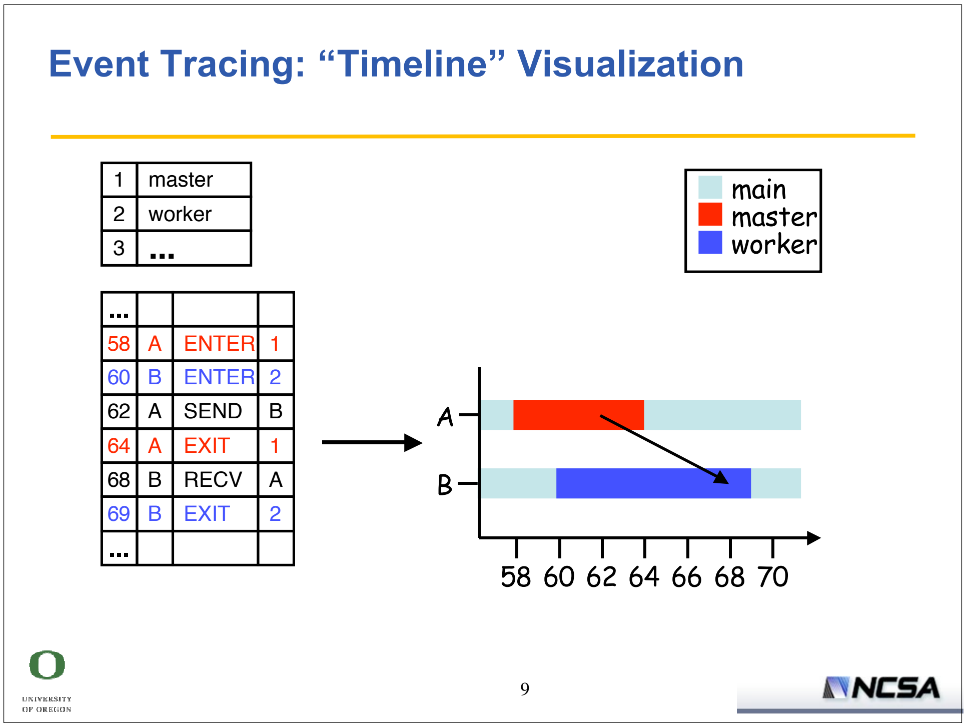
CPU B:

```
void worker {
  trace(ENTER, 2);
  ...
  recv(A, tag, buf);
  trace(RECV, A);
  ...
  trace(EXIT, 2);
}
```

Event definition

1	master
2	worker
3	...

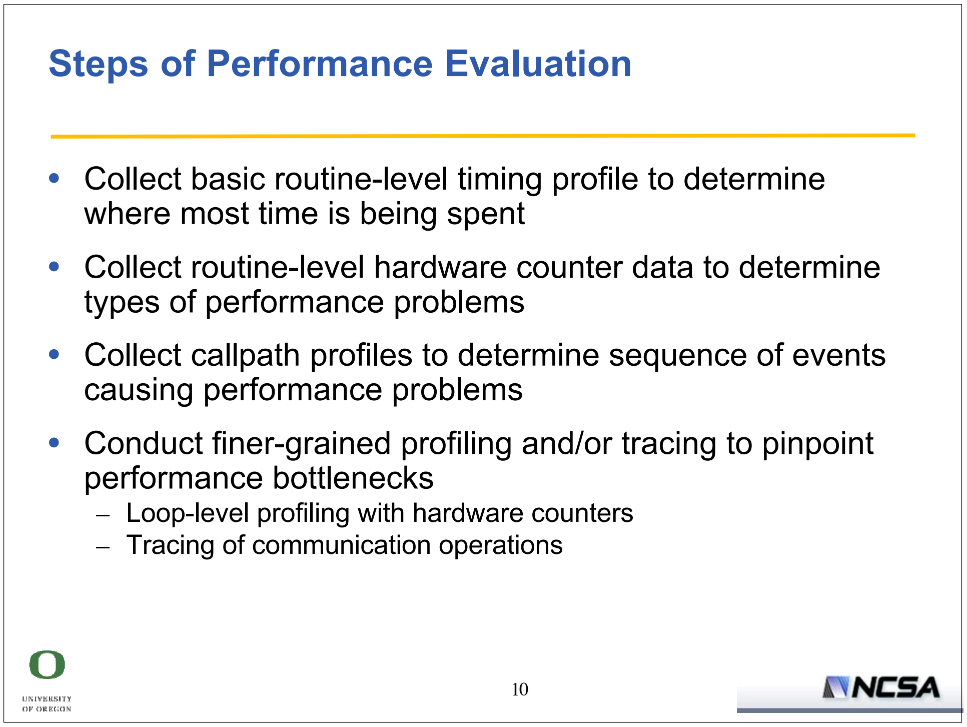
...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



Event Tracing: “Timeline” Visualization

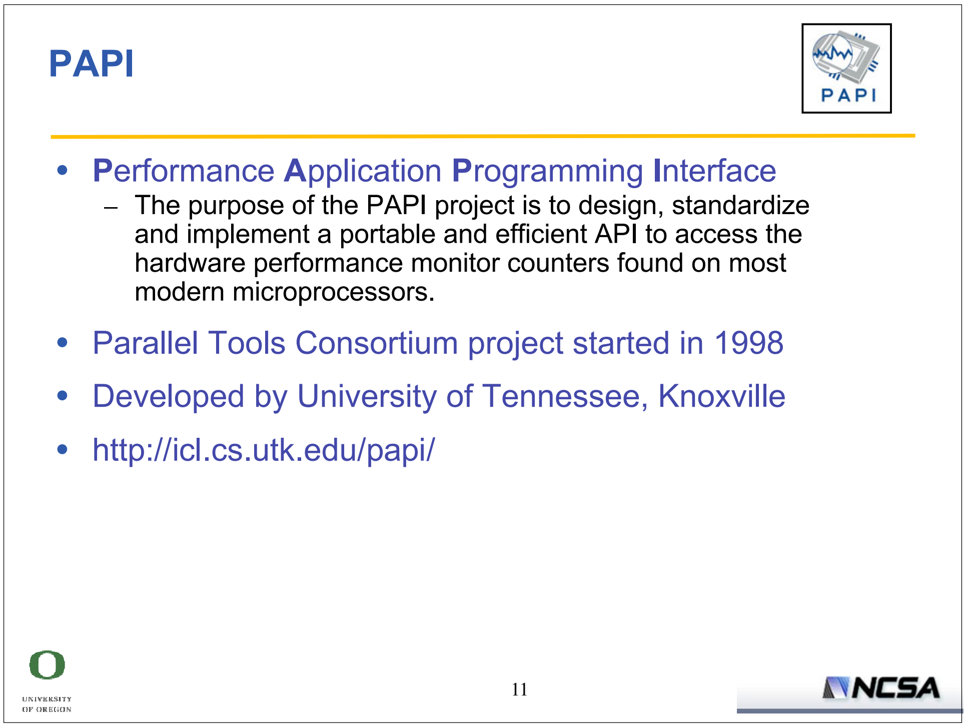
1	master
2	worker
3	...

...			
58	A	ENTER	1
60	B	ENTER	2
62	A	SEND	B
64	A	EXIT	1
68	B	RECV	A
69	B	EXIT	2
...			



Steps of Performance Evaluation

- Collect basic routine-level timing profile to determine where most time is being spent
- Collect routine-level hardware counter data to determine types of performance problems
- Collect callpath profiles to determine sequence of events causing performance problems
- Conduct finer-grained profiling and/or tracing to pinpoint performance bottlenecks
  - Loop-level profiling with hardware counters
  - Tracing of communication operations



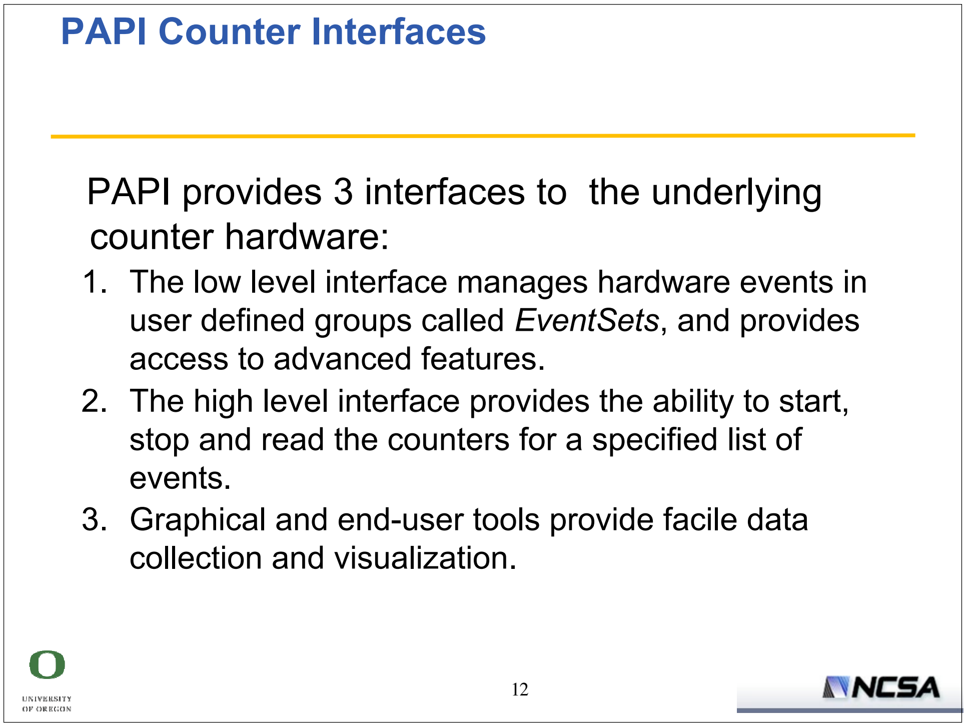
PAPI

- **P**erformance **A**pplication **P**rogramming **I**nterface
  - The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- Parallel Tools Consortium project started in 1998
- Developed by University of Tennessee, Knoxville
- http://icl.cs.utk.edu/papi/

UNIVERSITY OF OREGON

11

NCSA



PAPI Counter Interfaces

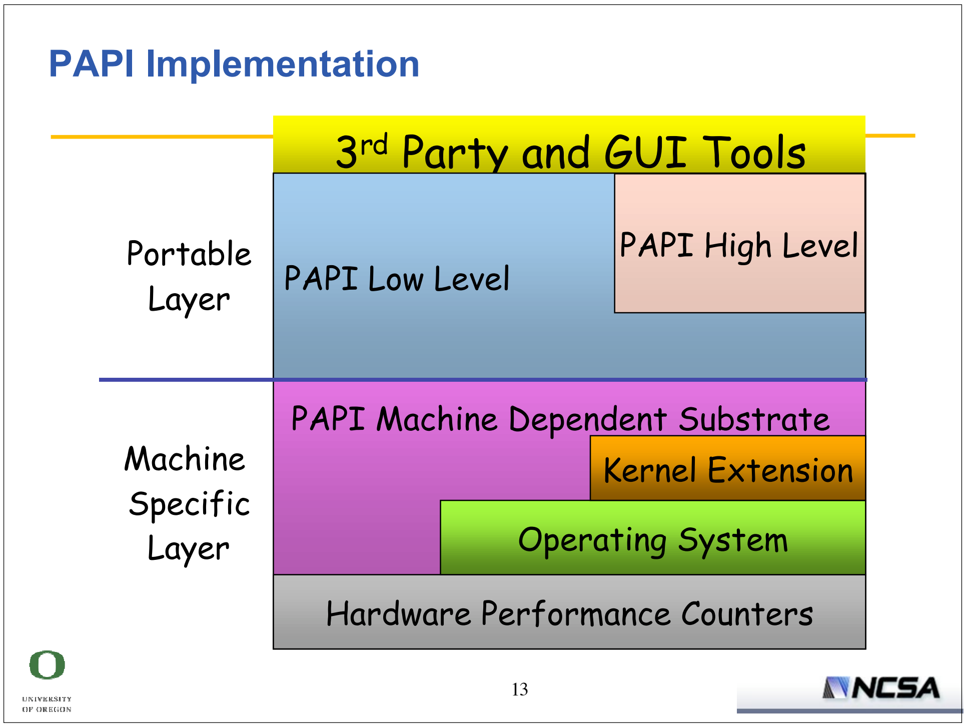
PAPI provides 3 interfaces to the underlying counter hardware:

1. The low level interface manages hardware events in user defined groups called *EventSets*, and provides access to advanced features.
2. The high level interface provides the ability to start, stop and read the counters for a specified list of events.
3. Graphical and end-user tools provide facile data collection and visualization.

UNIVERSITY OF OREGON

12

NCSA



PAPI Implementation

3rd Party and GUI Tools

Portable Layer

PAPI Low Level

PAPI High Level

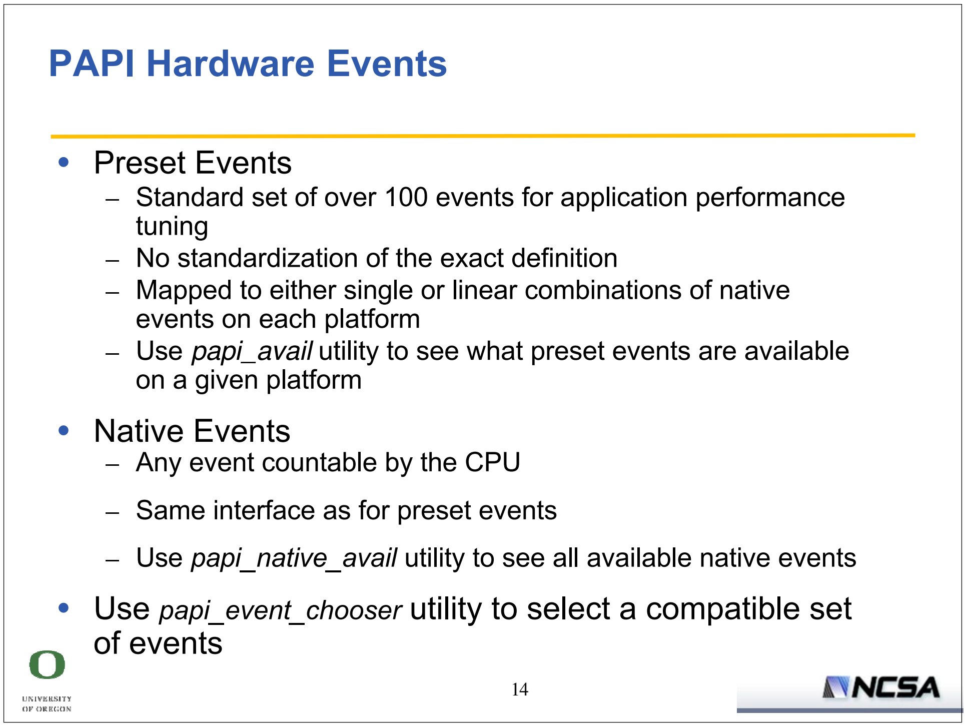
Machine Specific Layer

PAPI Machine Dependent Substrate

Kernel Extension

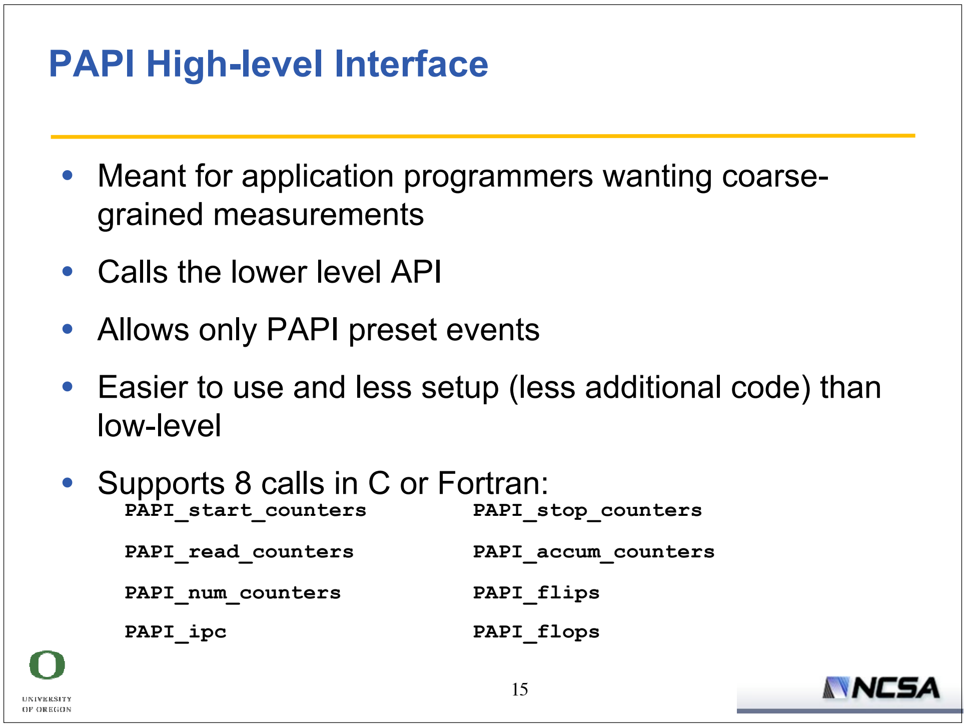
Operating System

Hardware Performance Counters



PAPI Hardware Events

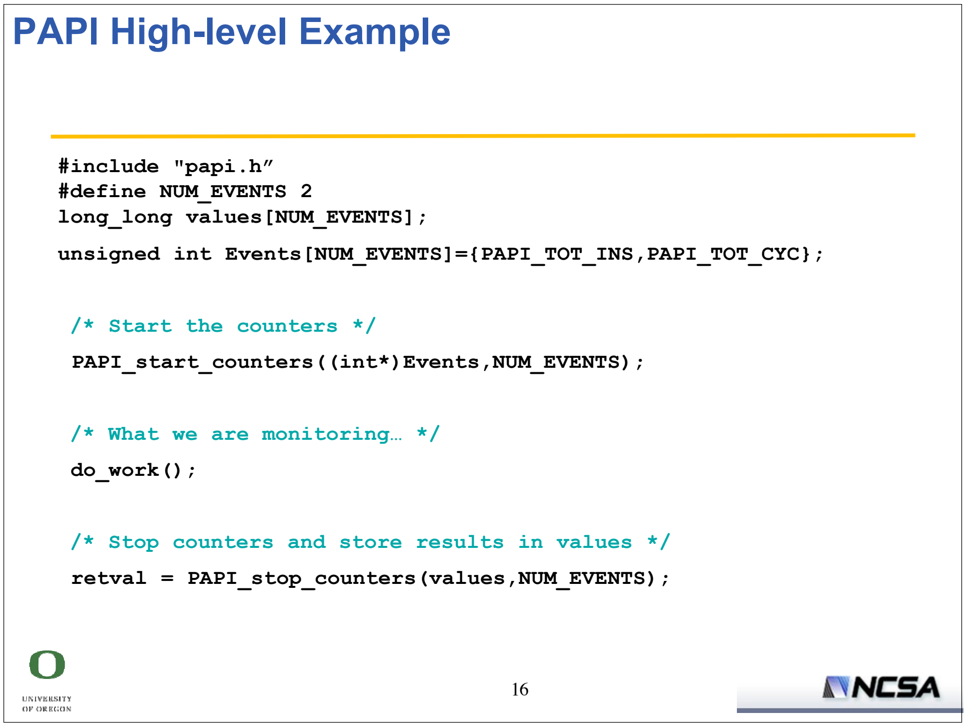
- Preset Events
  - Standard set of over 100 events for application performance tuning
  - No standardization of the exact definition
  - Mapped to either single or linear combinations of native events on each platform
  - Use *papi_avail* utility to see what preset events are available on a given platform
- Native Events
  - Any event countable by the CPU
  - Same interface as for preset events
  - Use *papi_native_avail* utility to see all available native events
- Use *papi_event_chooser* utility to select a compatible set of events



PAPI High-level Interface

- Meant for application programmers wanting coarse-grained measurements
- Calls the lower level API
- Allows only PAPI preset events
- Easier to use and less setup (less additional code) than low-level
- Supports 8 calls in C or Fortran:

```
PAPI_start_counters          PAPI_stop_counters
PAPI_read_counters           PAPI_accum_counters
PAPI_num_counters            PAPI_flips
PAPI_ipc                     PAPI_flops
```



PAPI High-level Example

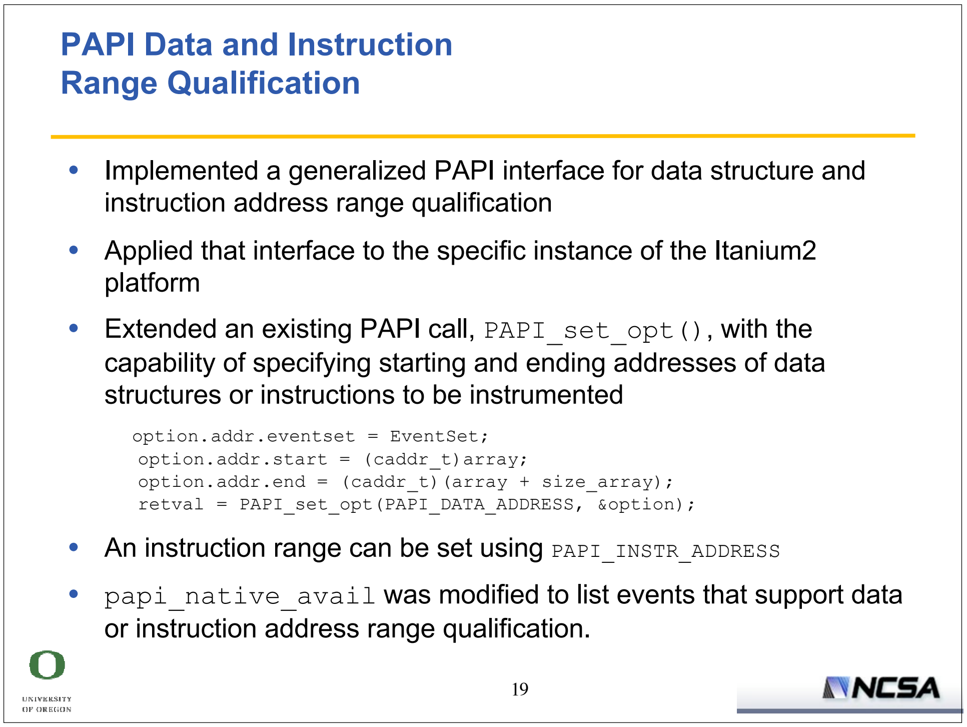
```
#include "papi.h”
#define NUM_EVENTS 2
long_long values[NUM_EVENTS];
unsigned int Events[NUM_EVENTS]={PAPI_TOT_INS,PAPI_TOT_CYC};

 /* Start the counters */
 PAPI_start_counters((int*)Events,NUM_EVENTS);

 /* What we are monitoring… */
 do_work();

 /* Stop counters and store results in values */
 retval = PAPI_stop_counters(values,NUM_EVENTS);
```

Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- Obtain information about the executable, the hardware, and the memory environment
- Multiplexing
- Callbacks on counter overflow
- Profiling
- About 60 functions

PAPI Low-level Example

```
#include "papi.h”
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC};
int EventSet;
long_long values[NUM_EVENTS];
/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* Add Flops and total cycles to the eventset */
retval = PAPI_add_events(EventSet,Events,NUM_EVENTS);
/* Start the counters */
retval = PAPI_start(EventSet);

do_work();  /* What we want to monitor*/

/*Stop counters and store results in values */
retval = PAPI_stop(EventSet,values);
```

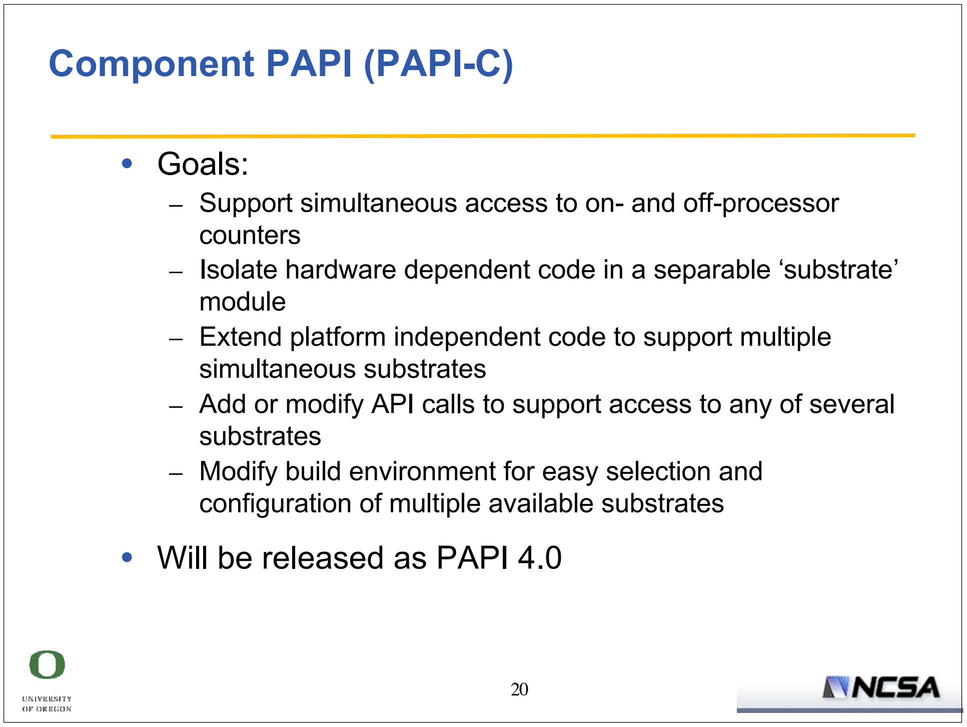


PAPI Data and Instruction Range Qualification

- Implemented a generalized PAPI interface for data structure and instruction address range qualification
- Applied that interface to the specific instance of the Itanium2 platform
- Extended an existing PAPI call, `PAPI_set_opt()`, with the capability of specifying starting and ending addresses of data structures or instructions to be instrumented

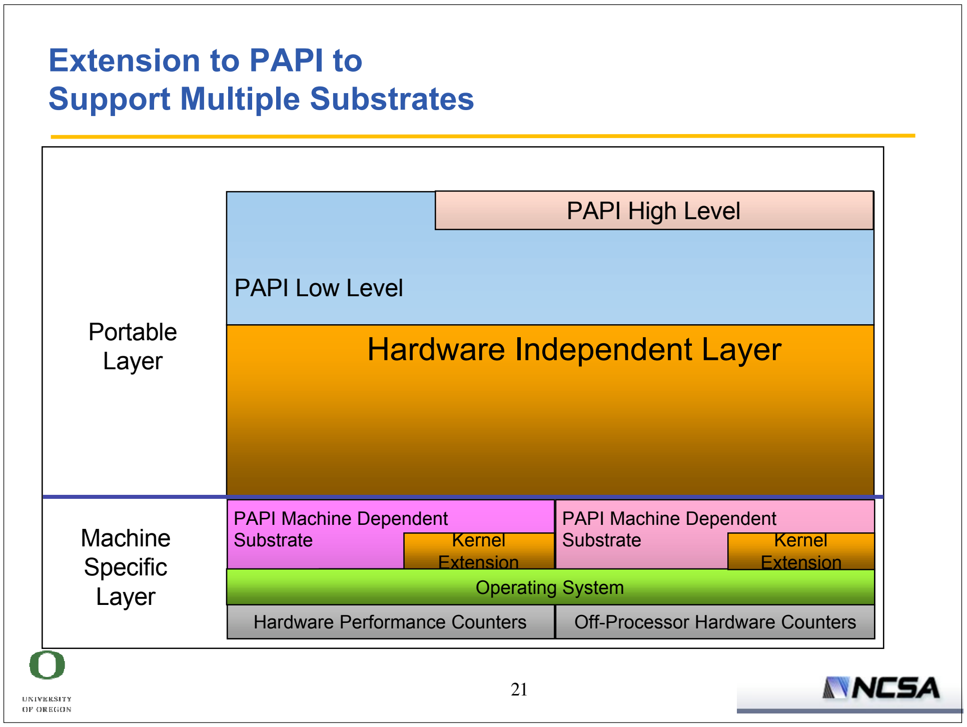
```
option.addr.eventset = EventSet;
 option.addr.start = (caddr_t)array;
 option.addr.end = (caddr_t)(array + size_array);
 retval = PAPI_set_opt(PAPI_DATA_ADDRESS, &option);
```

- An instruction range can be set using `PAPI_INSTR_ADDRESS`
- `papi_native_avail` was modified to list events that support data or instruction address range qualification.



Component PAPI (PAPI-C)

- Goals:
  - Support simultaneous access to on- and off-processor counters
  - Isolate hardware dependent code in a separable ‘substrate’ module
  - Extend platform independent code to support multiple simultaneous substrates
  - Add or modify API calls to support access to any of several substrates
  - Modify build environment for easy selection and configuration of multiple available substrates
- Will be released as PAPI 4.0



Extension to PAPI to Support Multiple Substrates

PAPI High Level

PAPI Low Level

Portable Layer

Hardware Independent Layer

Machine Specific Layer

PAPI Machine Dependent Substrate

Kernel Extension

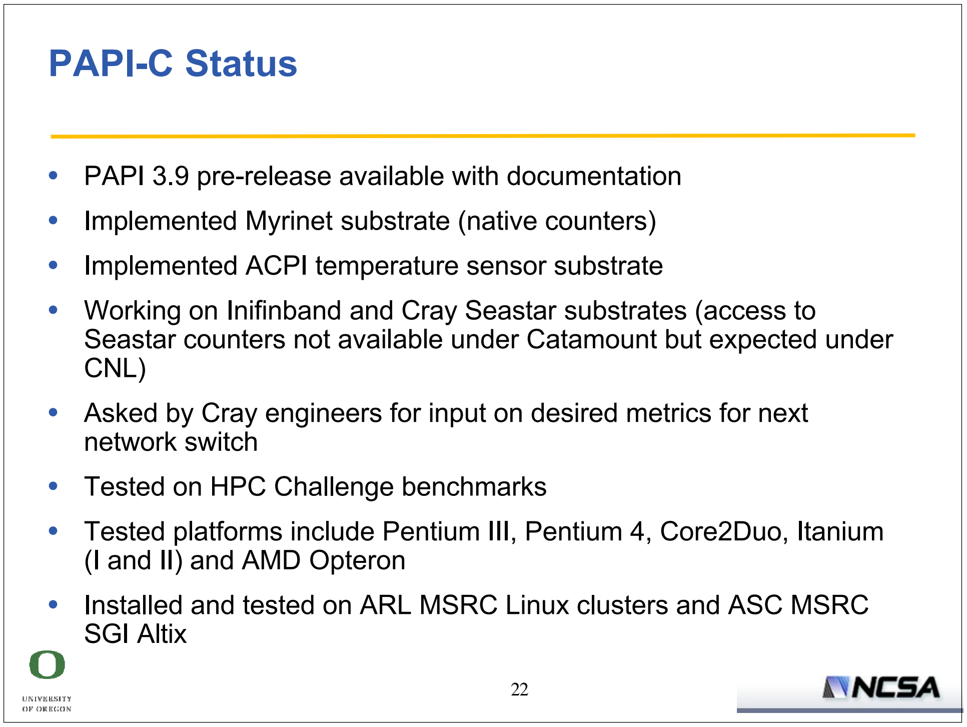
PAPI Machine Dependent Substrate

Kernel Extension

Operating System

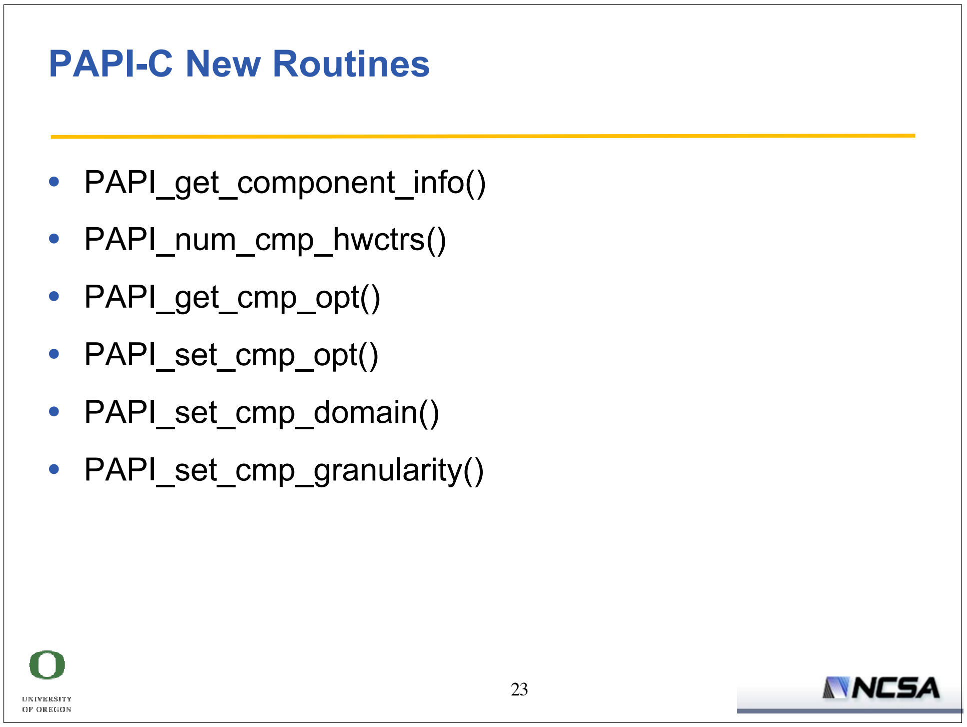
Hardware Performance Counters

Off-Processor Hardware Counters



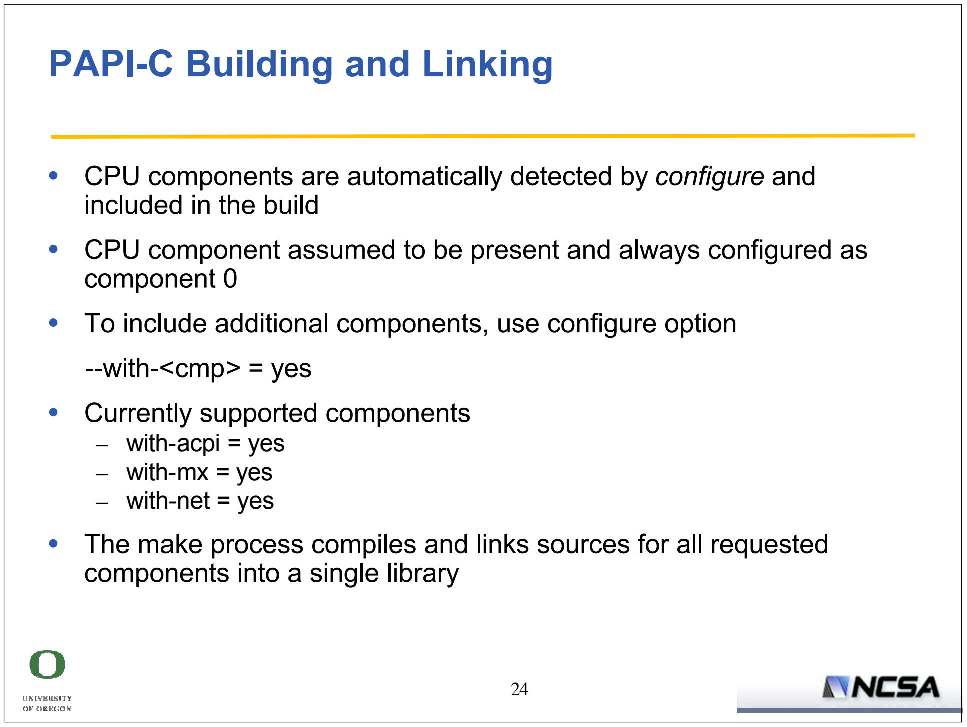
PAPI-C Status

- PAPI 3.9 pre-release available with documentation
- Implemented Myrinet substrate (native counters)
- Implemented ACPI temperature sensor substrate
- Working on Inifinband and Cray Seastar substrates (access to Seastar counters not available under Catamount but expected under CNL)
- Asked by Cray engineers for input on desired metrics for next network switch
- Tested on HPC Challenge benchmarks
- Tested platforms include Pentium III, Pentium 4, Core2Duo, Itanium (I and II) and AMD Opteron
- Installed and tested on ARL MSRC Linux clusters and ASC MSRC SGI Altix



PAPI-C New Routines

- PAPI_get_component_info()
- PAPI_num_cmp_hwctrs()
- PAPI_get_cmp_opt()
- PAPI_set_cmp_opt()
- PAPI_set_cmp_domain()
- PAPI_set_cmp_granularity()



PAPI-C Building and Linking

- CPU components are automatically detected by *configure* and included in the build
- CPU component assumed to be present and always configured as component 0
- To include additional components, use configure option

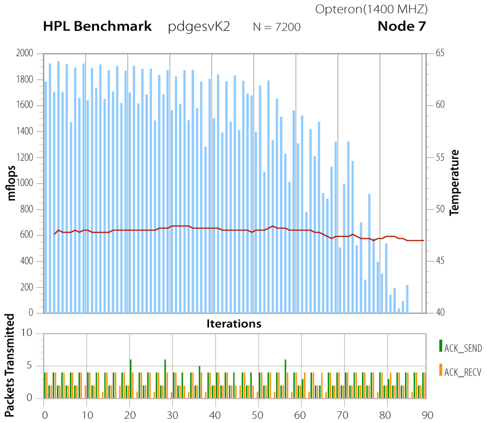
  --with-<cmp> = yes
- Currently supported components
  - with-acpi = yes
  - with-mx = yes
  - with-net = yes
- The make process compiles and links sources for all requested components into a single library

Myrinet MX Counters

```
LANAI_UPTIME
COUNTERS_UPTIME
BAD_CRC8
BAD_CRC32
UNSTRIPPED_ROUTE
PKT_DESC_INVALID
RECV_PKT_ERRORS
PKT_MISROUTED
DATA_SRC_UNKNOWN
DATA_BAD_ENDPT
DATA_ENDPT_CLOSED
DATA_BAD_SESSION
PUSH_BAD_WINDOW
PUSH_DUPLICATE
PUSH_OBSOLETE
PUSH_RACE_DRIVER
PUSH_BAD_SEND_HANDLE_MAGIC
PUSH_BAD_SRC_MAGIC
PULL_OBSOLETE
PULL_NOTIFY_OBSOLETE
PULL_RACE_DRIVER
ACK_BAD_TYPE
ACK_BAD_MAGIC
ACK_RESEND_RACE
LATE_ACKh
ACK_NACK_FRAMES_IN_PIPE
NACK_BAD_ENDPT
NACK_ENDPT_CLOSED
NACK_BAD_SESSION
NACK_BAD_RDMAWIN
NACK_EVENTQ_FULL
SEND_BAD_RDMAWIN
CONNECT_TIMEOUT
CONNECT_SRC_UNKNOWN
QUERY_BAD_MAGIC
QUERY_TIMED_OUT
QUERY_SRC_UNKNOWN
RAW_SENDS
RAW_RECEIVES
RAW_OVERSIZED_PACKETS
RAW_RECV_OVERRUN
RAW_DISABLED
CONNECT_SEND
CONNECT_RECV
ACK_SEND
ACK_RECV
PUSH_SEND
PUSH_RECV
QUERY_SEND
QUERY_RECV
REPLY_SEND
REPLY_RECV
QUERY_UNKNOWN
DATA_SEND_NULL
DATA_SEND_SMALL
DATA_SEND_MEDIUM
DATA_SEND_RNDV
DATA_SEND_PULL
DATA_RECV_NULL
DATA_RECV_SMALL_INLINE
DATA_RECV_SMALL_COPY
DATA_RECV_MEDIUM
DATA_RECV_RNDV
DATA_RECV_PULL
ETHER_SEND_UNICAST_CNT
ETHER_SEND_MULTICAST_CNT
ETHER_RECV_SMALL_CNT
ETHER_RECV_BIG_CNT
ETHER_OVERRUN
ETHER_OVERSIZED
DATA_RECV_NO_CREDITS
PACKETS_RESENT
PACKETS_DROPPED
MAPPER_ROUTES_UPDATE
ROUTE_DISPERSION
OUT_OF_SEND_HANDLES
OUT_OF_PULL_HANDLES
OUT_OF_PUSH_HANDLES
MEDIUM_CONT_RACE
CMD_TYPE_UNKNOWN
UREQ_TYPE_UNKNOWN
INTERRUPTS_OVERRUN
WAITING_FOR_INTERRUPT_DMA
WAITING_FOR_INTERRUPT_ACK
WAITING_FOR_INTERRUPT_TIMER
SLABS_RECYCLING
SLABS_PRESSURE
SLABS_STARVATION
OUT_OF_RDMA_HANDLES
EVENTQ_FULL
BUFFER_DROP
MEMORY_DROP
HARDWARE_FLOW_CONTROL
SIMULATED_PACKETS_LOST
LOGGING_FRAMES_DUMPED
WAKE_INTERRUPTS
AVERTED_WAKEUP_RACE
DMA_METADATA_RACE
```

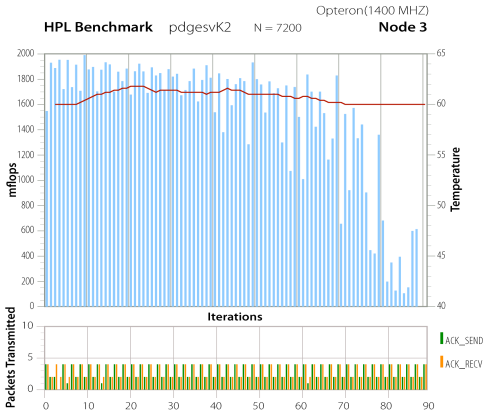
Multiple Measurements

- HPCC HPL benchmark on Opteron with 3 performance metrics:
  - FLOPS; Temperature; Network Sends/Receives
    - Temperature is from an on-chip thermal diode



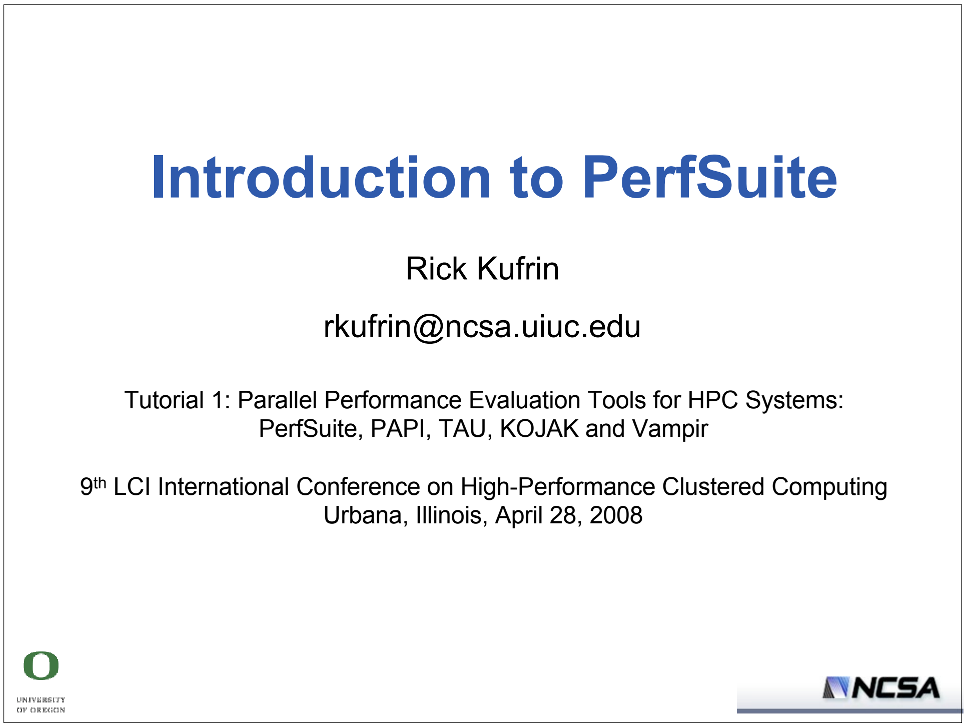
Multiple Measurements

- HPCC HPL benchmark on Opteron with 3 performance metrics:
  - FLOPS; Temperature; Network Sends/Receives
    - Temperature is from an on-chip thermal diode



Tools that use PAPI
Most users access PAPI from higher-level tools

- TAU (U Oregon) http://tau.uoregon.edu/
- HPCToolkit (Rice Univ) http://hipersoft.cs.rice.edu/hpctoolkit/
- KOJAK (UTK, FZ Juelich) http://icl.cs.utk.edu/kojak/
- PerfSuite (NCSA) http://perfsuite.ncsa.uiuc.edu/
- Titanium (UC Berkeley)
  http://www.cs.berkeley.edu/Research/Projects/titanium/
- SCALEA (Thomas Fahringer, U Innsbruck)
  http://www.par.univie.ac.at/project/scalea/
- Open|Speedshop (SGI) http://oss.sgi.com/projects/openspeedshop/
- SvPablo (UNC Renaissance Computing Institute)
  http://www.renci.unc.edu/Software/Pablo/pablo.htm



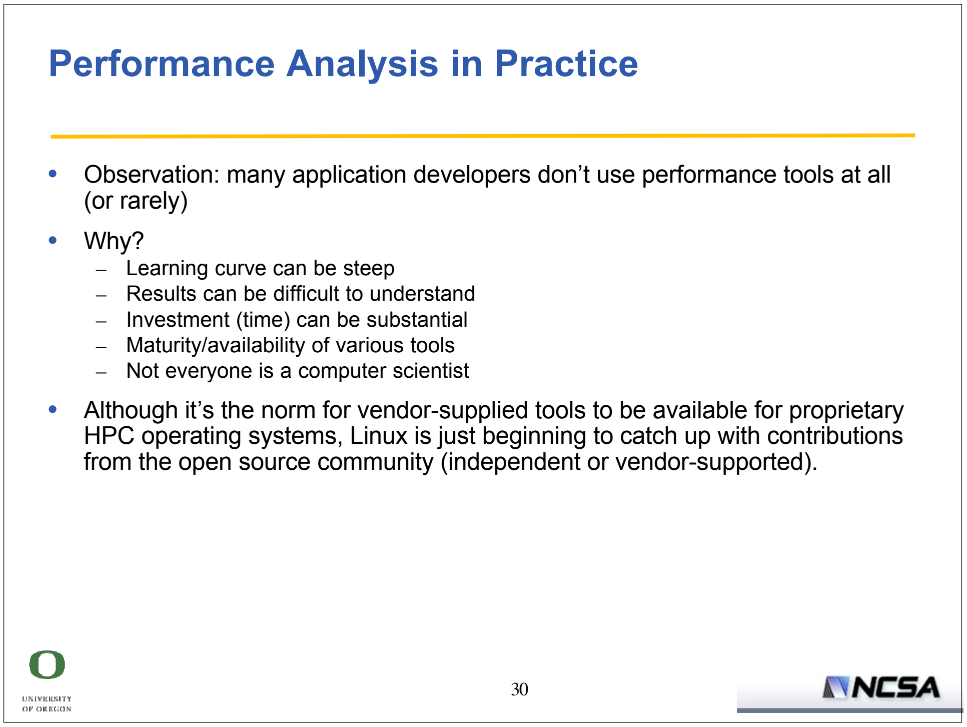
Introduction to PerfSuite

Rick Kufrin

rkufrin@ncsa.uiuc.edu

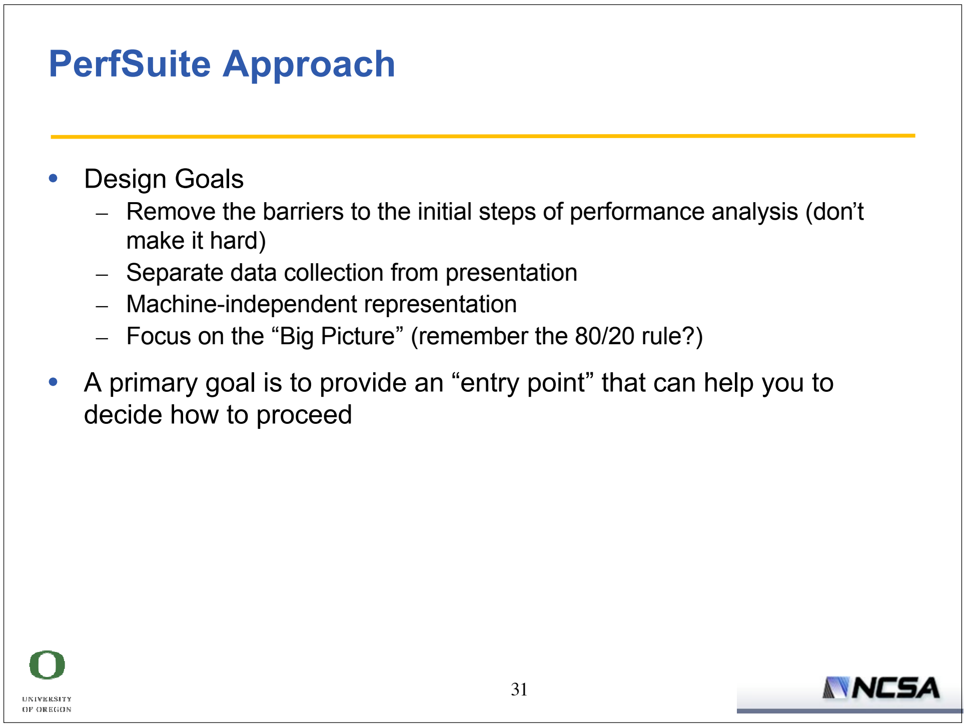
Tutorial 1: Parallel Performance Evaluation Tools for HPC Systems:
PerfSuite, PAPI, TAU, KOJAK and Vampir

9th LCI International Conference on High-Performance Clustered Computing
Urbana, Illinois, April 28, 2008



Performance Analysis in Practice

- Observation: many application developers don’t use performance tools at all (or rarely)
- Why?
  - Learning curve can be steep
  - Results can be difficult to understand
  - Investment (time) can be substantial
  - Maturity/availability of various tools
  - Not everyone is a computer scientist
- Although it’s the norm for vendor-supplied tools to be available for proprietary HPC operating systems, Linux is just beginning to catch up with contributions from the open source community (independent or vendor-supported).



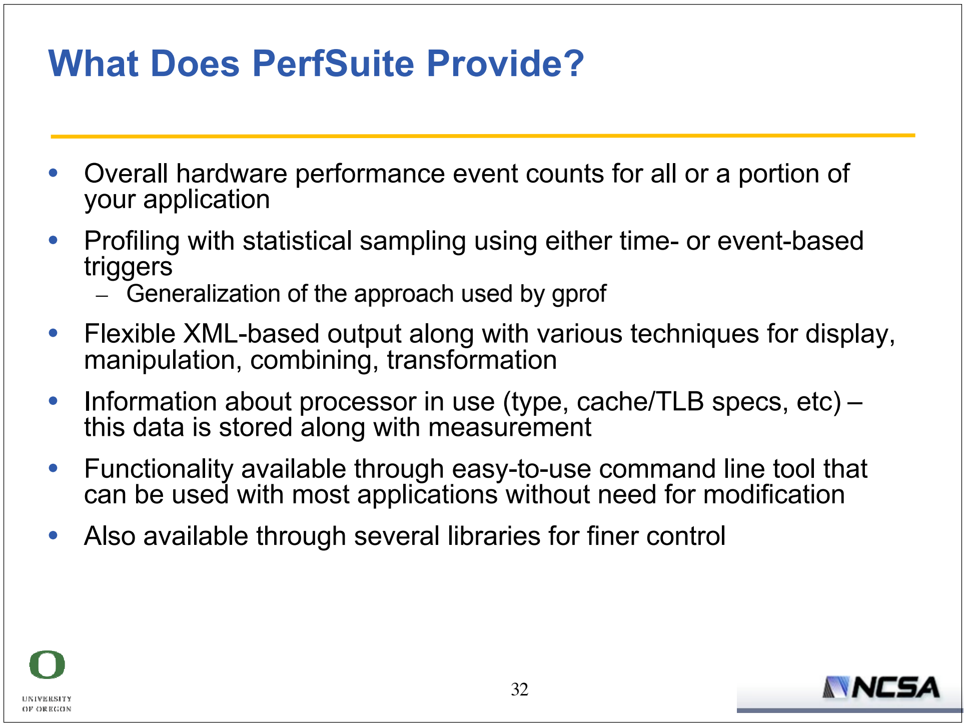
PerfSuite Approach

- Design Goals
  - Remove the barriers to the initial steps of performance analysis (don't make it hard)
  - Separate data collection from presentation
  - Machine-independent representation
  - Focus on the "Big Picture" (remember the 80/20 rule?)
- A primary goal is to provide an "entry point" that can help you to decide how to proceed

UNIVERSITY OF OREGON

31

NCSA



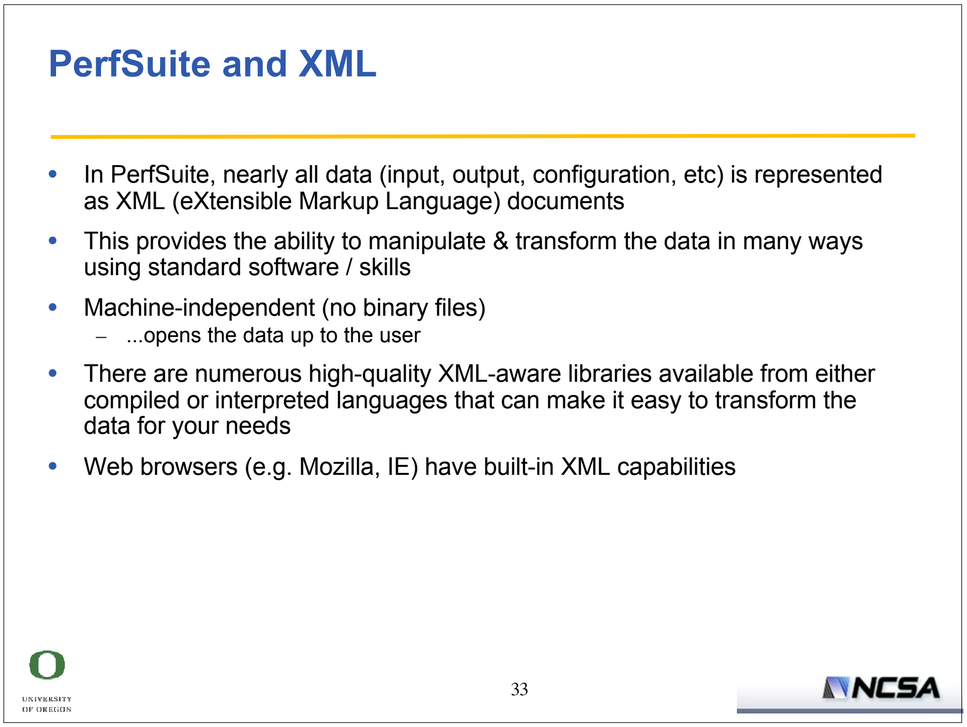
What Does PerfSuite Provide?

- Overall hardware performance event counts for all or a portion of your application
- Profiling with statistical sampling using either time- or event-based triggers
  - Generalization of the approach used by gprof
- Flexible XML-based output along with various techniques for display, manipulation, combining, transformation
- Information about processor in use (type, cache/TLB specs, etc) – this data is stored along with measurement
- Functionality available through easy-to-use command line tool that can be used with most applications without need for modification
- Also available through several libraries for finer control

UNIVERSITY OF OREGON

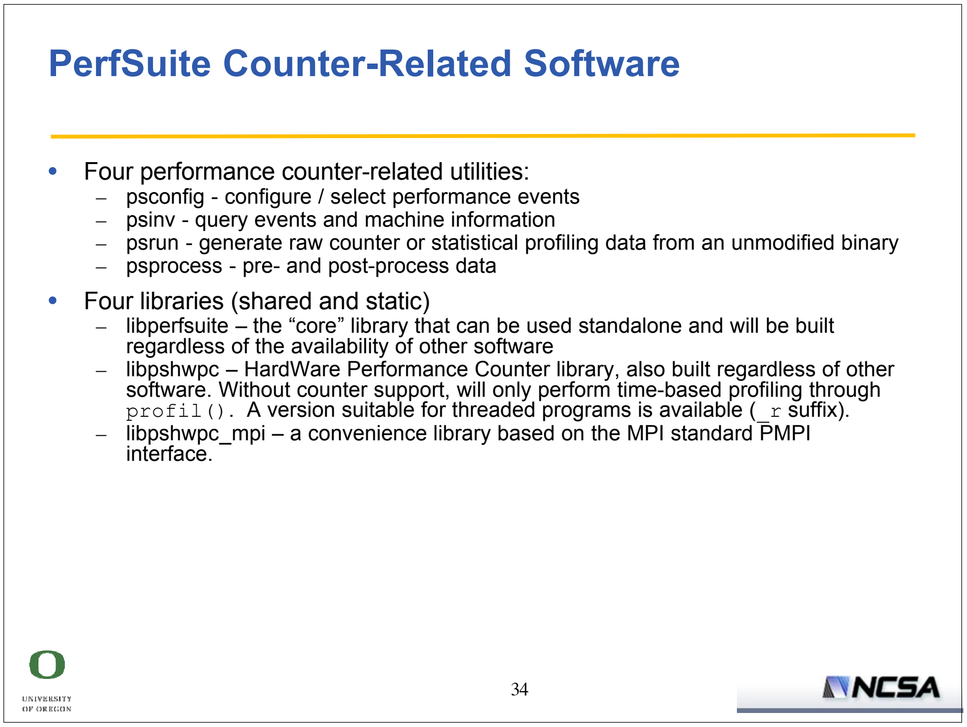
32

NCSA



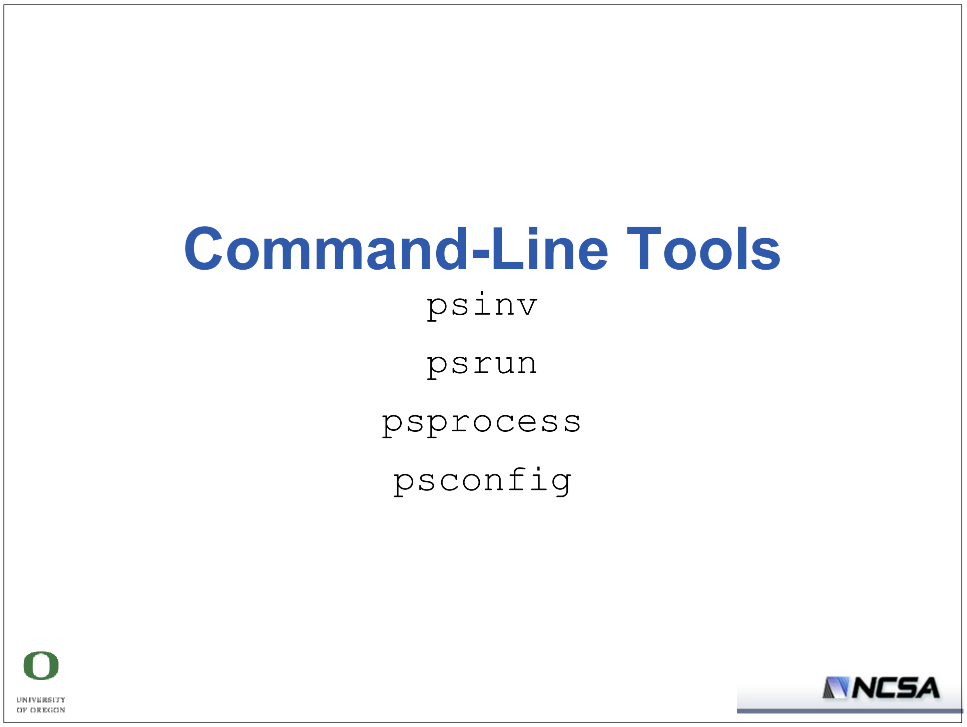
PerfSuite and XML

- In PerfSuite, nearly all data (input, output, configuration, etc) is represented as XML (eXtensible Markup Language) documents
- This provides the ability to manipulate & transform the data in many ways using standard software / skills
- Machine-independent (no binary files)
  - ...opens the data up to the user
- There are numerous high-quality XML-aware libraries available from either compiled or interpreted languages that can make it easy to transform the data for your needs
- Web browsers (e.g. Mozilla, IE) have built-in XML capabilities



PerfSuite Counter-Related Software

- Four performance counter-related utilities:
  - psconfig - configure / select performance events
  - psinv - query events and machine information
  - psrun - generate raw counter or statistical profiling data from an unmodified binary
  - psprocess - pre- and post-process data
- Four libraries (shared and static)
  - libperfsuite – the "core" library that can be used standalone and will be built regardless of the availability of other software
  - libpshwpc – HardWare Performance Counter library, also built regardless of other software. Without counter support, will only perform time-based profiling through `profil()`. A version suitable for threaded programs is available (`_r` suffix).
  - libpshwpc_mpi – a convenience library based on the MPI standard PMPI interface.



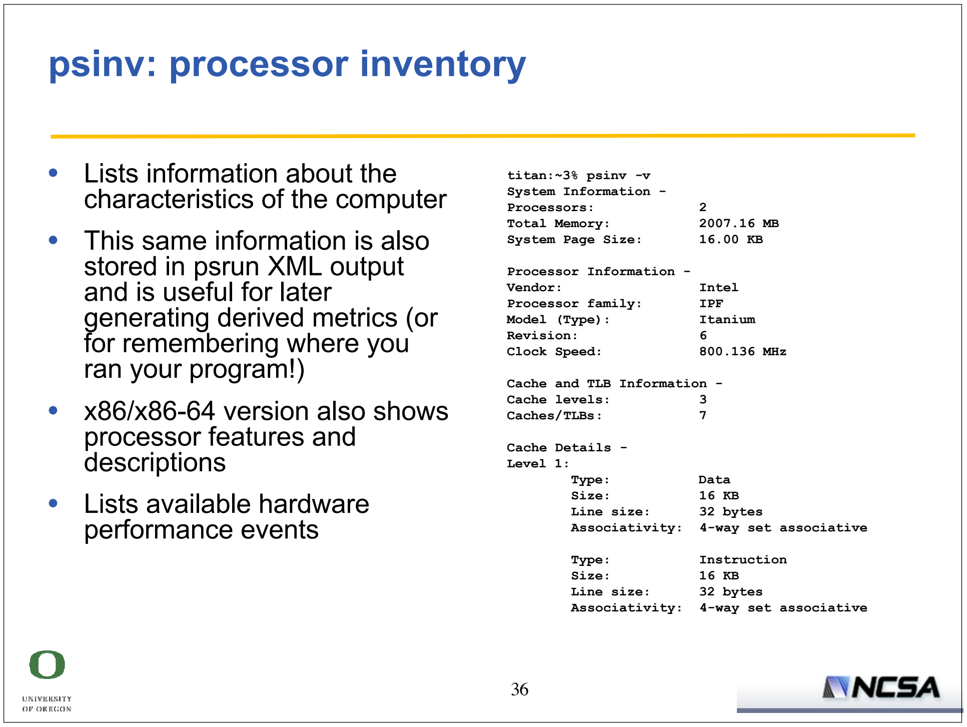
Command-Line Tools

`psinv`

`psrun`

`psprocess`

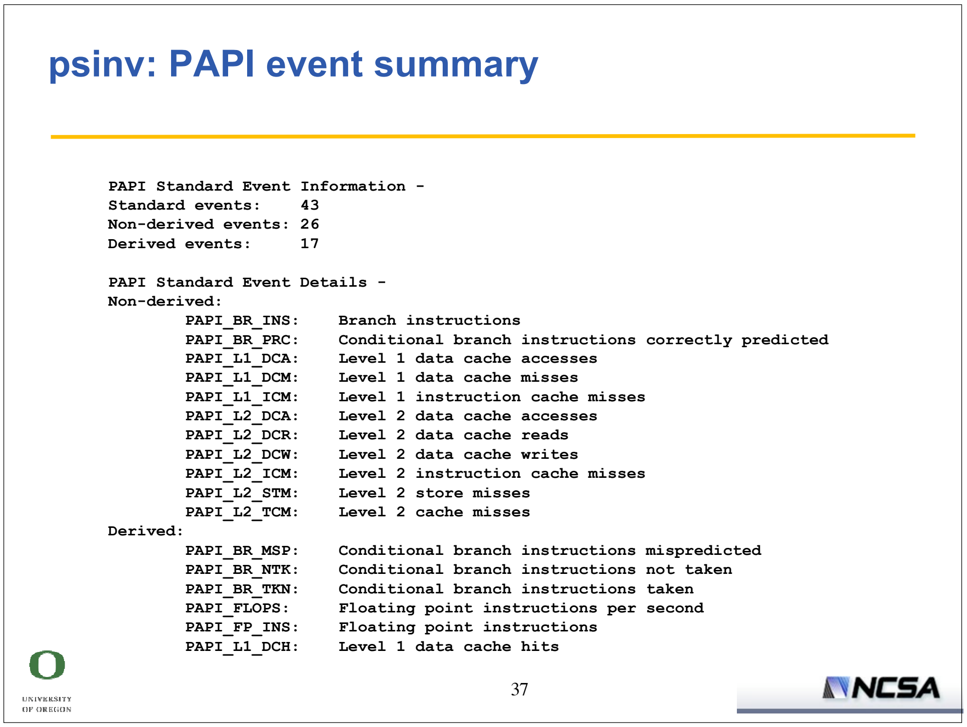
`psconfig`



psinv: processor inventory

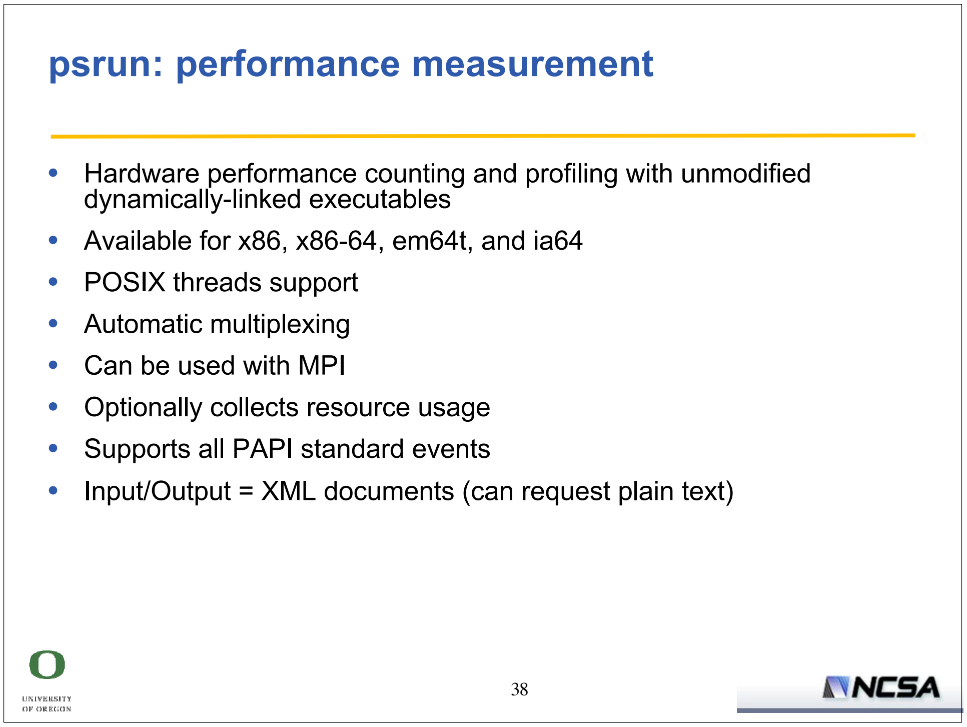
- Lists information about the characteristics of the computer
- This same information is also stored in psrun XML output and is useful for later generating derived metrics (or for remembering where you ran your program!)
- x86/x86-64 version also shows processor features and descriptions
- Lists available hardware performance events

```
titan:~3% psinv -v
System Information -
Processors:             2
Total Memory:           2007.16 MB
System Page Size:       16.00 KB

Processor Information -
Vendor:                 Intel
Processor family:       IPF
Model (Type):           Itanium
Revision:               6
Clock Speed:            800.136 MHz

Cache and TLB Information -
Cache levels:           3
Caches/TLBs:            7

Cache Details -
Level 1:
        Type:           Data
        Size:           16 KB
        Line size:      32 bytes
        Associativity:  4-way set associative

        Type:           Instruction
        Size:           16 KB
        Line size:      32 bytes
        Associativity:  4-way set associative
```



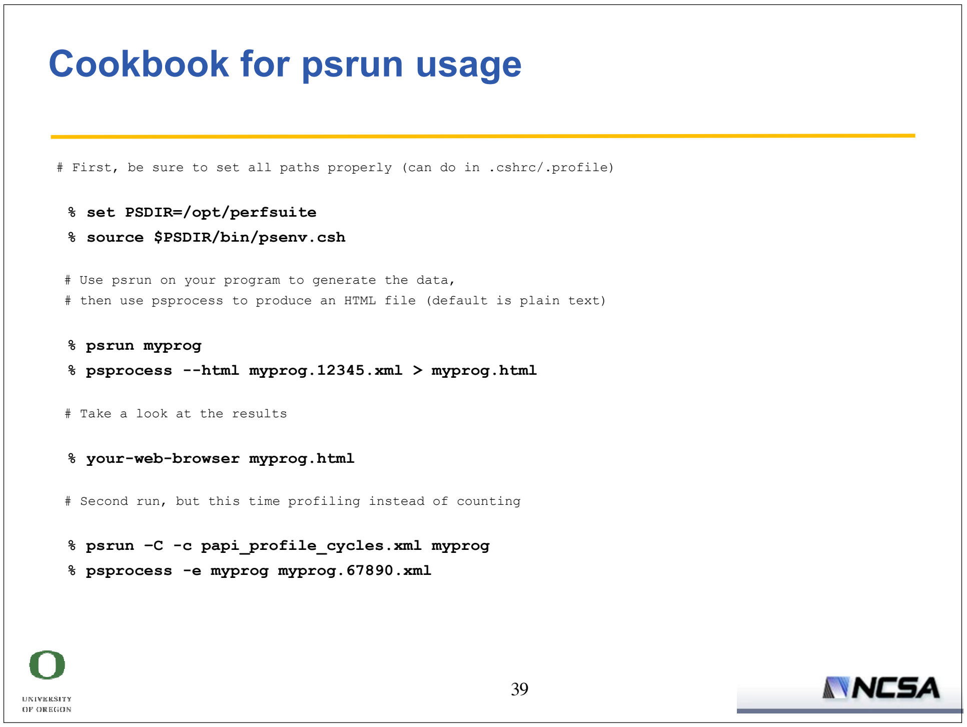
psinv: PAPI event summary

```
PAPI Standard Event Information -
Standard events:    43
Non-derived events: 26
Derived events:     17

PAPI Standard Event Details -
Non-derived:
        PAPI_BR_INS:    Branch instructions
        PAPI_BR_PRC:    Conditional branch instructions correctly predicted
        PAPI_L1_DCA:    Level 1 data cache accesses
        PAPI_L1_DCM:    Level 1 data cache misses
        PAPI_L1_ICM:    Level 1 instruction cache misses
        PAPI_L2_DCA:    Level 2 data cache accesses
        PAPI_L2_DCR:    Level 2 data cache reads
        PAPI_L2_DCW:    Level 2 data cache writes
        PAPI_L2_ICM:    Level 2 instruction cache misses
        PAPI_L2_STM:    Level 2 store misses
        PAPI_L2_TCM:    Level 2 cache misses
Derived:
        PAPI_BR_MSP:    Conditional branch instructions mispredicted
        PAPI_BR_NTK:    Conditional branch instructions not taken
        PAPI_BR_TKN:    Conditional branch instructions taken
        PAPI_FLOPS:     Floating point instructions per second
        PAPI_FP_INS:    Floating point instructions
        PAPI_L1_DCH:    Level 1 data cache hits
```



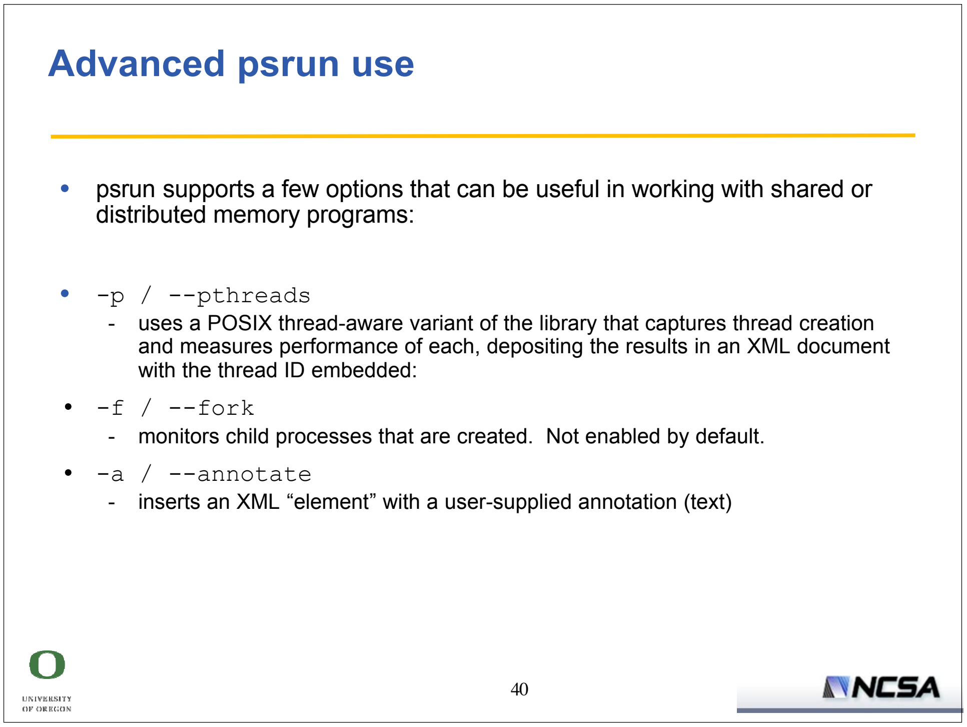
psrun: performance measurement

- Hardware performance counting and profiling with unmodified dynamically-linked executables
- Available for x86, x86-64, em64t, and ia64
- POSIX threads support
- Automatic multiplexing
- Can be used with MPI
- Optionally collects resource usage
- Supports all PAPI standard events
- Input/Output = XML documents (can request plain text)



Cookbook for psrun usage

```
# First, be sure to set all paths properly (can do in .cshrc/.profile)

 % set PSDIR=/opt/perfsuite
 % source $PSDIR/bin/psenv.csh

 # Use psrun on your program to generate the data,
 # then use psprocess to produce an HTML file (default is plain text)

 % psrun myprog
 % psprocess --html myprog.12345.xml > myprog.html

 # Take a look at the results

 % your-web-browser myprog.html

 # Second run, but this time profiling instead of counting

 % psrun –C -c papi_profile_cycles.xml myprog
 % psprocess -e myprog myprog.67890.xml
```



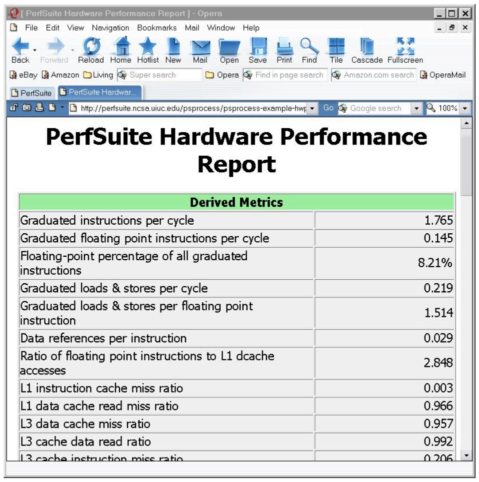
Advanced psrun use

- psrun supports a few options that can be useful in working with shared or distributed memory programs:

- `-p / --pthreads`
  - uses a POSIX thread-aware variant of the library that captures thread creation and measures performance of each, depositing the results in an XML document with the thread ID embedded:
- `-f / --fork`
  - monitors child processes that are created. Not enabled by default.
- `-a / --annotate`
  - inserts an XML “element” with a user-supplied annotation (text)

psprocess: post-process results

- This style of output is customizable by you.
- By default, the information it contains and its visual appearance are based on PerfSuite-provided defaults, but these can be easily replaced to suit your needs.
- This output is generated by psprocess using XML Transformations. The stylesheet is in the `share/perfsuite/xml/pshwpc` subdirectory, with a “xsl” file extension

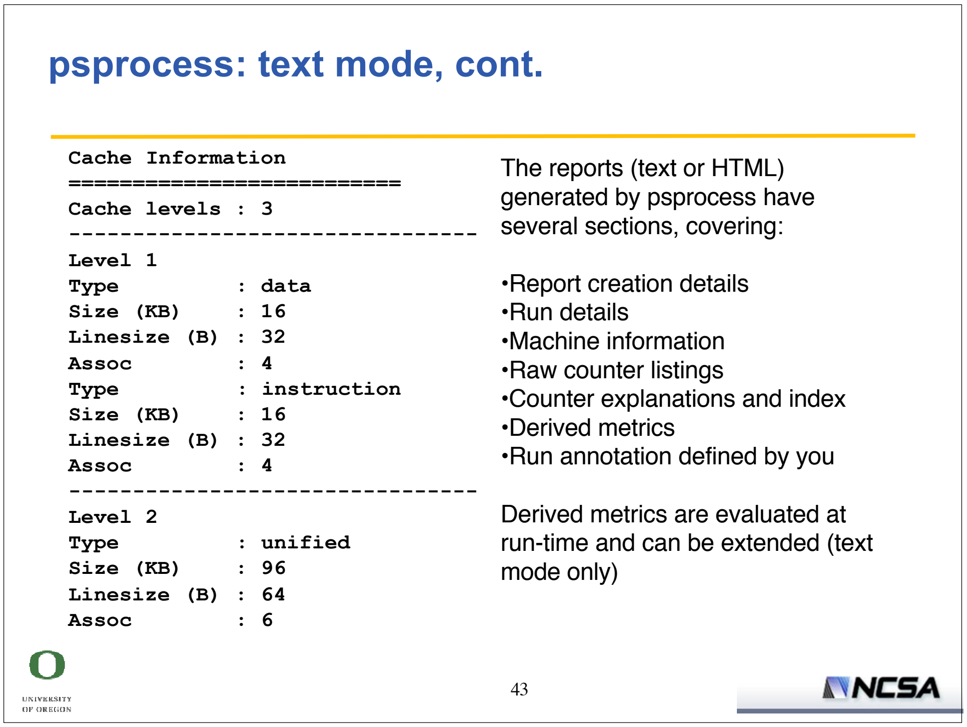

[PerfSuite Hardware Performance Report] - Opera
File Edit View Navigation Bookmarks Mail Window Help
Back Forward Reload Home Hotlist New Mail Open Save Print Find Tile Cascade Fullscreen
eBay Amazon Living Super search Opera Find in page search Amazon.com search OperaMail
PerfSuite PerfSuite Hardwar...
http://perfsuite.ncsa.uiuc.edu/psprocess/psprocess-example-hwp
Go Google search 100%

PerfSuite Hardware Performance Report

Derived Metrics	
Graduated instructions per cycle	1.765
Graduated floating point instructions per cycle	0.145
Floating-point percentage of all graduated instructions	8.21%
Graduated loads & stores per cycle	0.219
Graduated loads & stores per floating point instruction	1.514
Data references per instruction	0.029
Ratio of floating point instructions to L1 dcache accesses	2.848
L1 instruction cache miss ratio	0.003
L1 data cache read miss ratio	0.966
L3 data cache miss ratio	0.957
L3 cache data read ratio	0.992
L3 cache instruction miss ratio	0.206

psprocess: text mode (default)

```
PerfSuite Hardware Performance Summary Report
Version       : 1.0
Created       : Mon Dec 30 11:31:53 AM Central Standard Time 2002
Generator     : psprocess 0.5
XML Source    : /u/ncsa/anyuser/performance/psrun-ia64.xml

Execution Information
===========================
Date          : Sun Dec 15 21:01:20 2002
Host          : user01

Processor and System Information
===========================
Node CPUs     : 2
Vendor        : Intel
Family        : IPF
Model         : Itanium
CPU Revision  : 6
Clock (MHz)   : 800.136
Memory (MB)   : 2007.16
Pagesize (KB) : 16
```



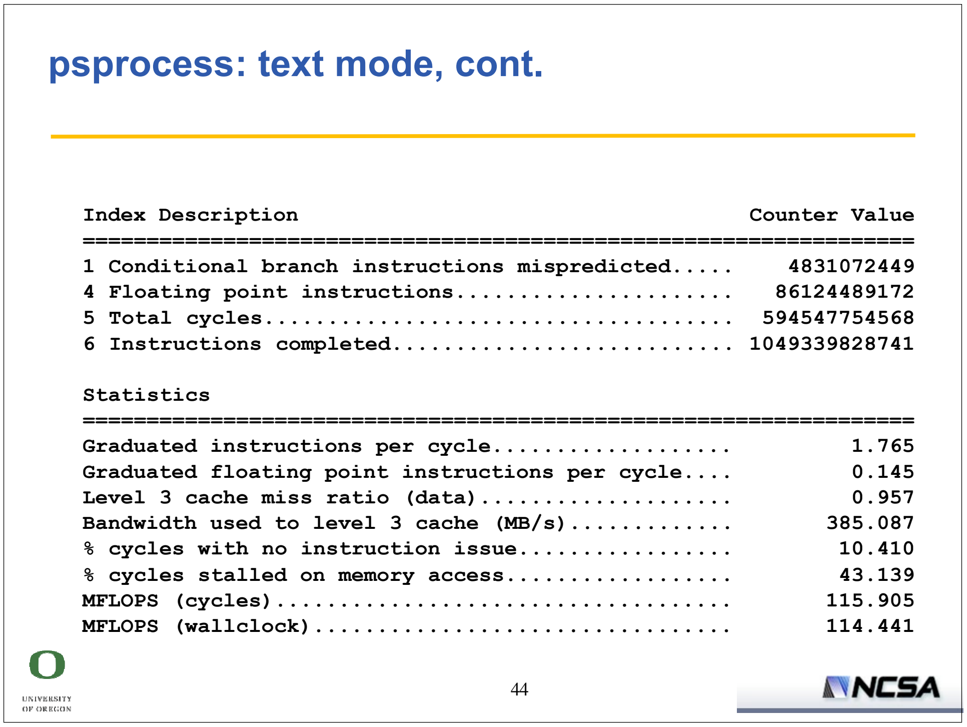
psprocess: text mode, cont.

```
Cache Information
==========================
Cache levels : 3
--------------------------------
Level 1
Type         : data
Size (KB)    : 16
Linesize (B) : 32
Assoc        : 4
Type         : instruction
Size (KB)    : 16
Linesize (B) : 32
Assoc        : 4
--------------------------------
Level 2
Type         : unified
Size (KB)    : 96
Linesize (B) : 64
Assoc        : 6
```

The reports (text or HTML) generated by psprocess have several sections, covering:

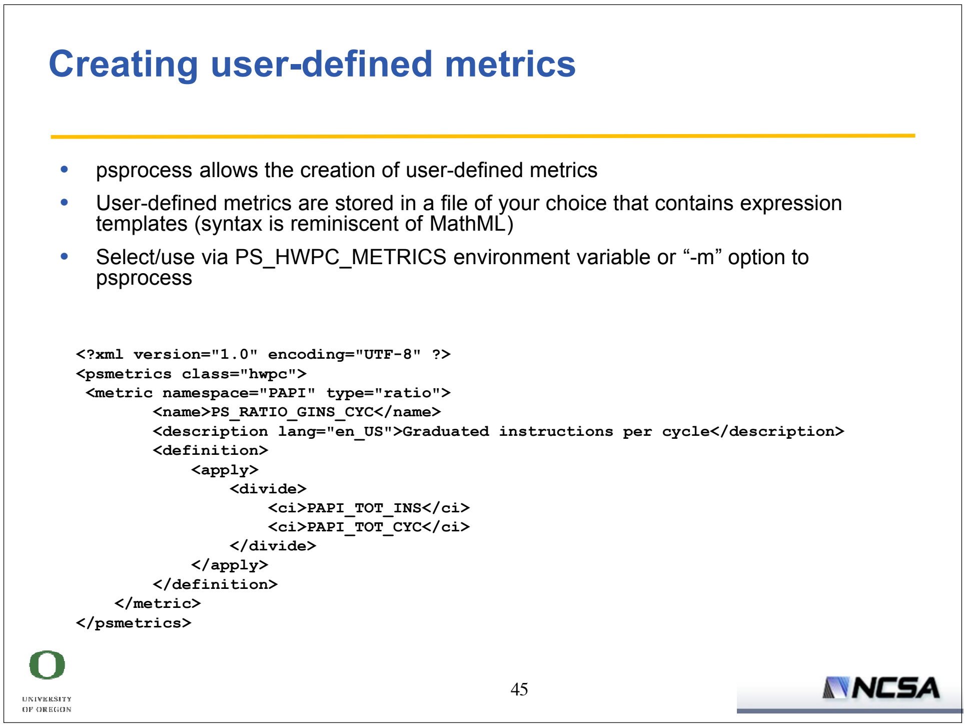
- Report creation details
- Run details
- Machine information
- Raw counter listings
- Counter explanations and index
- Derived metrics
- Run annotation defined by you

Derived metrics are evaluated at run-time and can be extended (text mode only)



psprocess: text mode, cont.

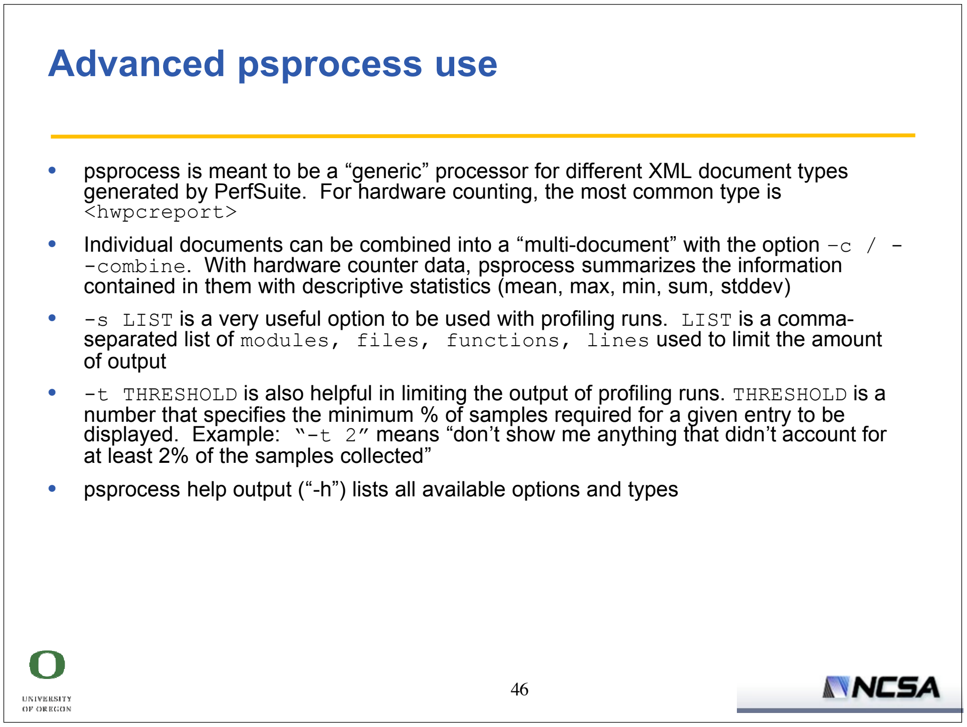
```
Index Description                                           Counter Value
=================================================================
1 Conditional branch instructions mispredicted.....      4831072449
4 Floating point instructions......................     86124489172
5 Total cycles....................................     594547754568
6 Instructions completed..........................    1049339828741

Statistics
=================================================================
Graduated instructions per cycle..................              1.765
Graduated floating point instructions per cycle....             0.145
Level 3 cache miss ratio (data)...................              0.957
Bandwidth used to level 3 cache (MB/s).............           385.087
% cycles with no instruction issue................             10.410
% cycles stalled on memory access.................             43.139
MFLOPS (cycles)...................................            115.905
MFLOPS (wallclock)................................            114.441
```



Creating user-defined metrics

- psprocess allows the creation of user-defined metrics
- User-defined metrics are stored in a file of your choice that contains expression templates (syntax is reminiscent of MathML)
- Select/use via PS_HWPC_METRICS environment variable or “-m” option to psprocess

```
<?xml version="1.0" encoding="UTF-8" ?>
<psmetrics class="hwpc">
 <metric namespace="PAPI" type="ratio">
        <name>PS_RATIO_GINS_CYC</name>
        <description lang="en_US">Graduated instructions per cycle</description>
        <definition>
            <apply>
                <divide>
                    <ci>PAPI_TOT_INS</ci>
                    <ci>PAPI_TOT_CYC</ci>
                </divide>
            </apply>
        </definition>
    </metric>
</psmetrics>
```



Advanced psprocess use

- psprocess is meant to be a “generic” processor for different XML document types generated by PerfSuite. For hardware counting, the most common type is `<hwpcreport>`
- Individual documents can be combined into a “multi-document” with the option `-c / --combine`. With hardware counter data, psprocess summarizes the information contained in them with descriptive statistics (mean, max, min, sum, stddev)
- `-s LIST` is a very useful option to be used with profiling runs. `LIST` is a comma-separated list of `modules, files, functions, lines` used to limit the amount of output
- `-t THRESHOLD` is also helpful in limiting the output of profiling runs. `THRESHOLD` is a number that specifies the minimum % of samples required for a given entry to be displayed. Example: “`-t 2`” means “don’t show me anything that didn’t account for at least 2% of the samples collected”
- psprocess help output (“-h”) lists all available options and types

Configuring PerfSuite

- All PerfSuite runs are configured according to an XML document that specifies what is to be measured
  - if you don’t specify a custom configuration, a default is used
- A custom configuration document (file) is supplied in one of two ways
  - psrun option “-c *filename*”
  - PS_HWPC_CONFIG environment variable, which can be set to *filename*
- Creating new configuration files is easy, and can be done with either a text editor or the tool “psconfig”

Example configuration

```
<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_eventlist class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_BR_MSP" />
  <ps_hwpc_event type="preset" name="PAPI_BR_PRC" />
  <ps_hwpc_event type="preset" name="PAPI_BR_TKN" />
  <ps_hwpc_event type="preset" name="PAPI_FP_INS" />
  <ps_hwpc_event type="preset" name="PAPI_TOT_CYC" />
  <ps_hwpc_event type="preset" name="PAPI_TOT_INS" />
  <ps_hwpc_event type="preset" name="PAPI_L1_DCA" />
  <ps_hwpc_event type="preset" name="PAPI_L1_DCM" />
  <ps_hwpc_event type="preset" name="PAPI_L1_ICR" />
  <ps_hwpc_event type="preset" name="PAPI_L1_TCM" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCA" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCM" />
</ps_hwpc_eventlist>
```

You can edit this file like any text file

The XML document root element “ps_hwpc_eventlist” indicates this configuration is to be used for aggregate counting (not profiling)

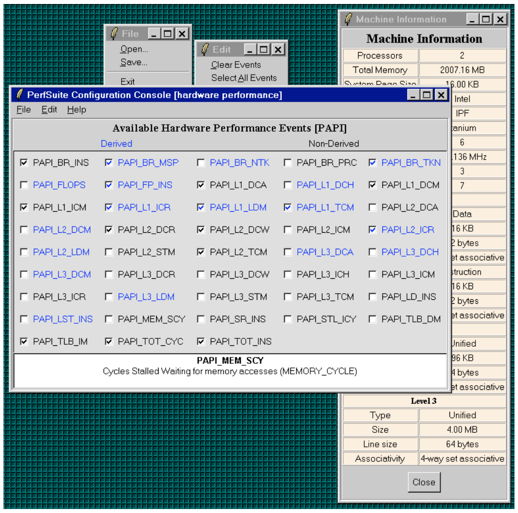
Configuring for profiling

- Setting up for profiling is similar to counting - all you have to do is modify the XML configuration document:
- The XML document “root element” is now <ps_hwpc_profile>, not <ps_hwpc_eventlist>
- You can supply an optional “threshold”, or sampling rate
- Only one event is allowed in the document
- psconfig does not yet support profiling, need to edit by hand

```
<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_profile class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_BR_MSP" threshold="100000" />
</ps_hwpc_profile>
```

psconfig: graphical configuration

- Graphical user interface makes it easy to select events
- Can read in or write out valid XML documents to be used by psrun
- Provides text description of events with mouse click
- Searching capabilities
- Profiling not yet supported



Available Hardware Performance Events [PAPI]

Derived — Non-Derived

☑ PAPI_BR_INS	☑ PAPI_BR_MSP	☐ PAPI_BR_NTK	☐ PAPI_BR_PRC	☑ PAPI_BR_TKN
☐ PAPI_FLOPS	☑ PAPI_FP_INS	☑ PAPI_L1_DCA	☐ PAPI_L1_DCH	☑ PAPI_L1_DCM
☑ PAPI_L1_ICM	☑ PAPI_L1_ICR	☑ PAPI_L1_LDM	☑ PAPI_L1_TCM	☐ PAPI_L2_DCA
☐ PAPI_L2_DCM	☑ PAPI_L2_DCR	☑ PAPI_L2_DCW	☐ PAPI_L2_ICM	☑ PAPI_L2_ICR
☐ PAPI_L2_LDM	☐ PAPI_L2_STM	☑ PAPI_L2_TCM	☐ PAPI_L3_DCA	☐ PAPI_L3_DCH
☐ PAPI_L3_DCM	☐ PAPI_L3_DCR	☐ PAPI_L3_DCW	☐ PAPI_L3_ICH	☐ PAPI_L3_ICM
☐ PAPI_L3_ICR	☐ PAPI_L3_LDM	☐ PAPI_L3_STM	☐ PAPI_L3_TCM	☐ PAPI_LD_INS
☐ PAPI_LST_INS	☐ PAPI_MEM_SCY	☐ PAPI_SR_INS	☐ PAPI_STL_ICY	☐ PAPI_TLB_DM
☑ PAPI_TLB_IM	☑ PAPI_TOT_CYC	☑ PAPI_TOT_INS		

PAPI_MEM_SCY

Cycles Stalled Waiting for memory accesses (MEMORY_CYCLE)

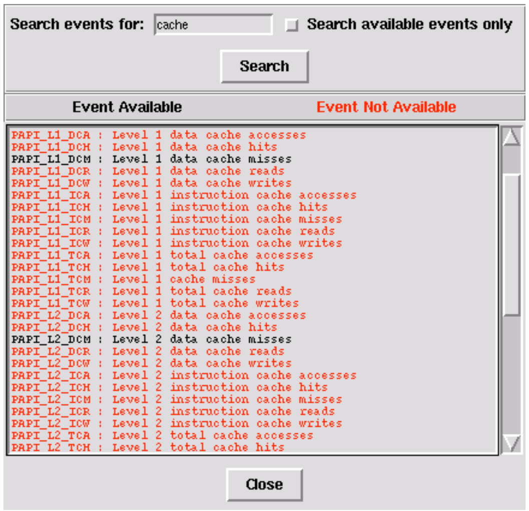
Machine Information

Processors	2
Total Memory	2007.16 MB
System Page Size	16.00 KB
	Intel
	IPF
	anium
	6
	136 MHz
	3
	7
	Data
	16 KB
	2 bytes
	et associative
	struction
	16 KB
	2 bytes
	et associative
	Unified
	96 KB
	4 bytes
	et associative
Level 3	
Type	Unified
Size	4.00 MB
Line size	64 bytes
Associativity	4-way set associative

Close

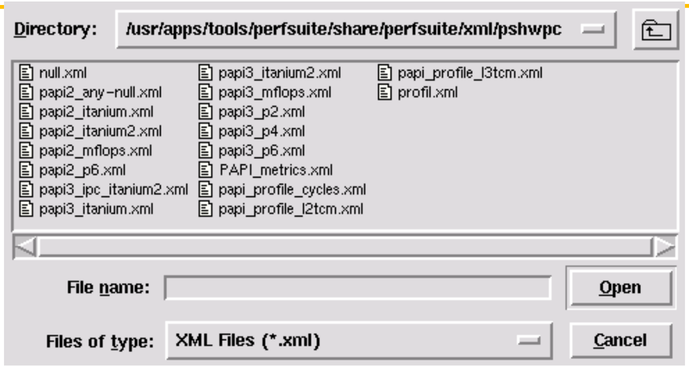
Searching events with psconfig

- Selecting "Edit", "Search Events…" brings up a window like this that allows you to search for keywords
- Can restrict the search to only events available on your computer
- The search is based on the event's description, not it's standard event name (PAPI_TOT_CYC)



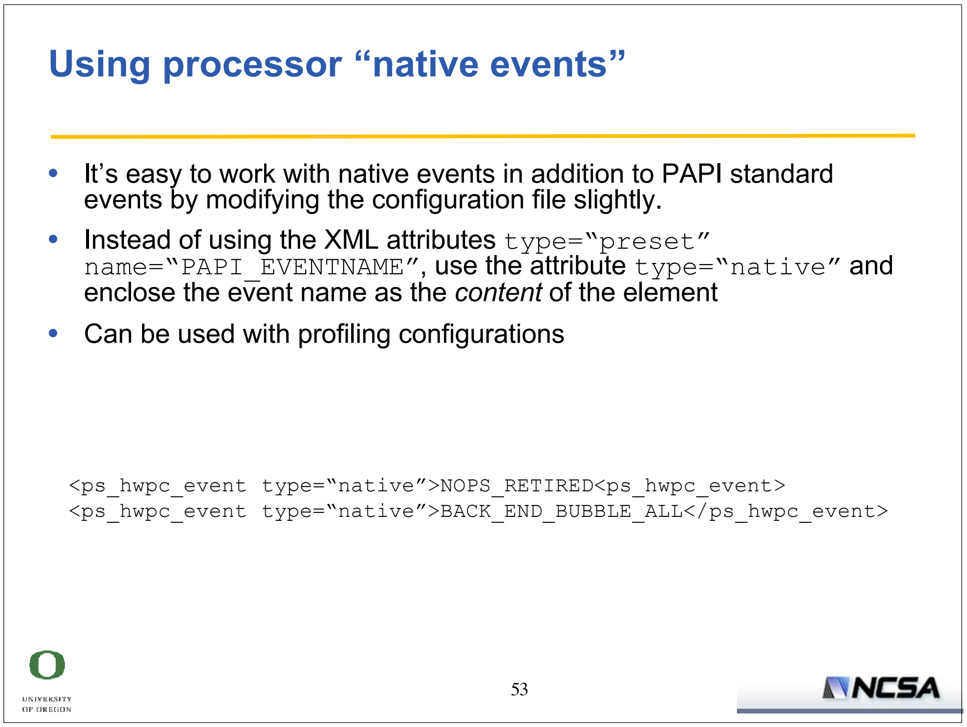
```
PAPI_L1_DCA : Level 1 data cache accesses
PAPI_L1_DCH : Level 1 data cache hits
PAPI_L1_DCM : Level 1 data cache misses
PAPI_L1_DCR : Level 1 data cache reads
PAPI_L1_DCW : Level 1 data cache writes
PAPI_L1_ICA : Level 1 instruction cache accesses
PAPI_L1_ICH : Level 1 instruction cache hits
PAPI_L1_ICM : Level 1 instruction cache misses
PAPI_L1_ICR : Level 1 instruction cache reads
PAPI_L1_ICW : Level 1 instruction cache writes
PAPI_L1_TCA : Level 1 total cache accesses
PAPI_L1_TCH : Level 1 total cache hits
PAPI_L1_TCM : Level 1 cache misses
PAPI_L1_TCR : Level 1 total cache reads
PAPI_L1_TCW : Level 1 total cache writes
PAPI_L2_DCA : Level 2 data cache accesses
PAPI_L2_DCH : Level 2 data cache hits
PAPI_L2_DCM : Level 2 data cache misses
PAPI_L2_DCR : Level 2 data cache reads
PAPI_L2_DCW : Level 2 data cache writes
PAPI_L2_ICA : Level 2 instruction cache accesses
PAPI_L2_ICH : Level 2 instruction cache hits
PAPI_L2_ICM : Level 2 instruction cache misses
PAPI_L2_ICR : Level 2 instruction cache reads
PAPI_L2_ICW : Level 2 instruction cache writes
PAPI_L2_TCA : Level 2 total cache accesses
PAPI_L2_TCH : Level 2 total cache hits
```

Browsing default event configurations



null.xml
papi2_any-null.xml
papi2_itanium.xml
papi2_itanium2.xml
papi2_mflops.xml
papi2_p6.xml
papi3_ipc_itanium2.xml
papi3_itanium.xml
papi3_itanium2.xml
papi3_mflops.xml
papi3_p2.xml
papi3_p4.xml
papi3_p6.xml
PAPI_metrics.xml
papi_profile_cycles.xml
papi_profile_l2tcm.xml
papi_profile_l3tcm.xml
profil.xml

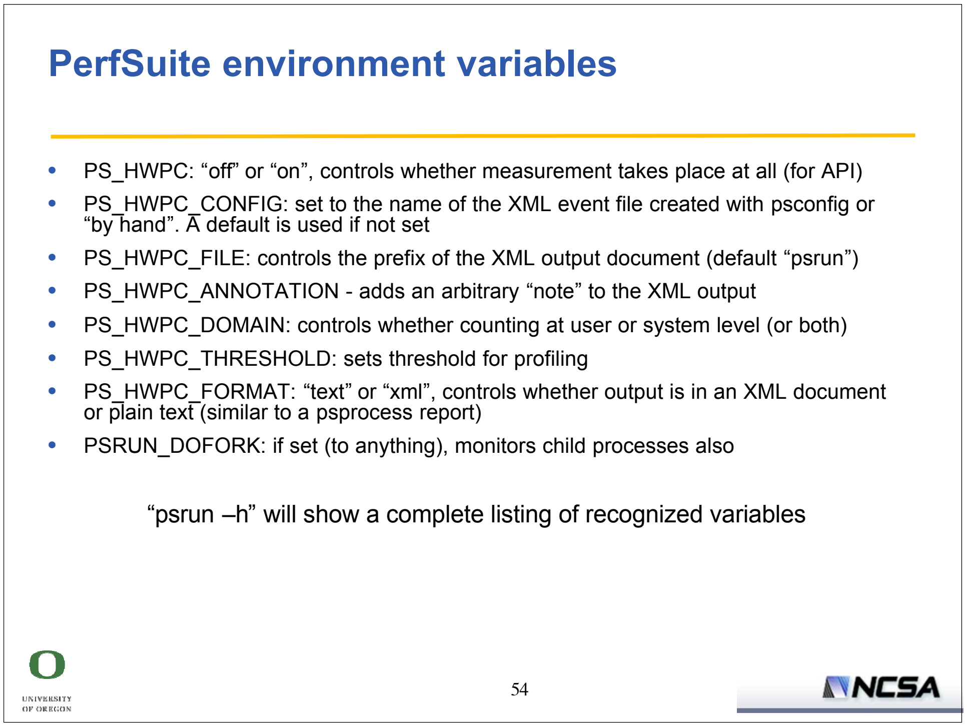
- Selecting "File", "Default Hardware Event Configurations…" brings up the directory with pre-selected configuration documents
- Opening one of them will show you which events will be used
- You can base custom configuration files using these as a start



Using processor “native events”

- It’s easy to work with native events in addition to PAPI standard events by modifying the configuration file slightly.
- Instead of using the XML attributes `type=“preset” name=“PAPI_EVENTNAME”`, use the attribute `type=“native”` and enclose the event name as the *content* of the element
- Can be used with profiling configurations

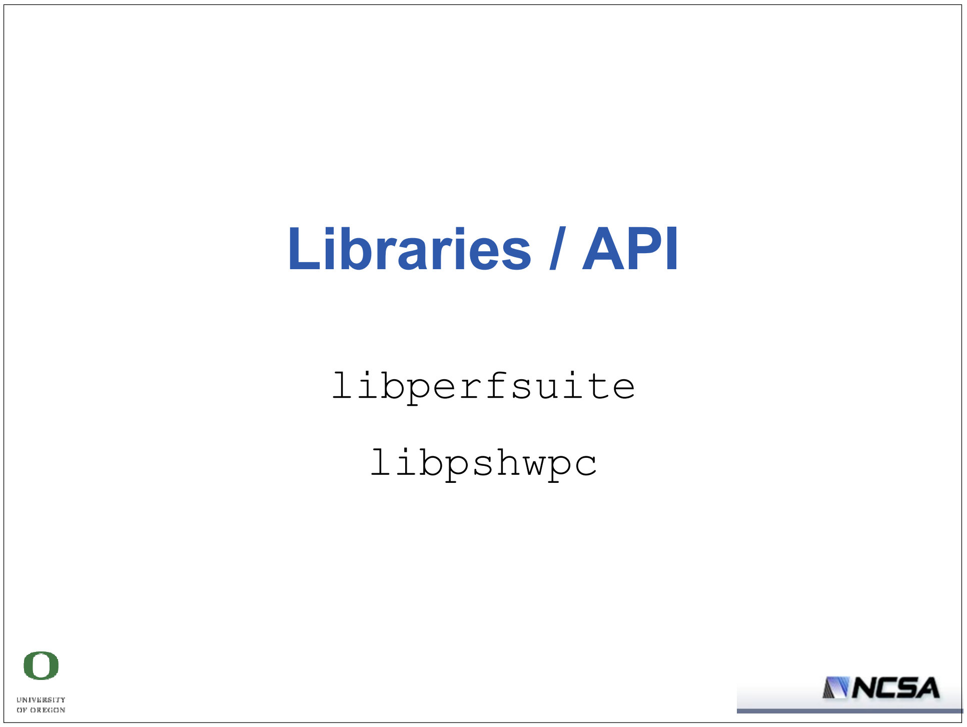
```
<ps_hwpc_event type=“native”>NOPS_RETIRED<ps_hwpc_event>
<ps_hwpc_event type=“native”>BACK_END_BUBBLE_ALL</ps_hwpc_event>
```



PerfSuite environment variables

- PS_HWPC: “off” or “on”, controls whether measurement takes place at all (for API)
- PS_HWPC_CONFIG: set to the name of the XML event file created with psconfig or “by hand”. A default is used if not set
- PS_HWPC_FILE: controls the prefix of the XML output document (default “psrun”)
- PS_HWPC_ANNOTATION - adds an arbitrary “note” to the XML output
- PS_HWPC_DOMAIN: controls whether counting at user or system level (or both)
- PS_HWPC_THRESHOLD: sets threshold for profiling
- PS_HWPC_FORMAT: “text” or “xml”, controls whether output is in an XML document or plain text (similar to a psprocess report)
- PSRUN_DOFORK: if set (to anything), monitors child processes also

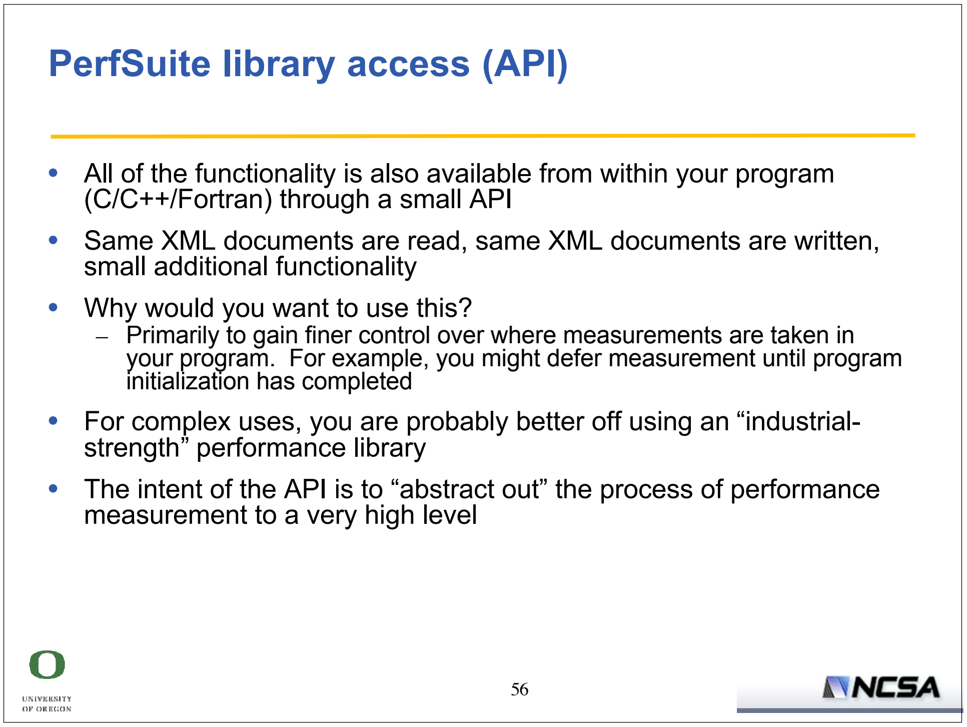
“psrun –h” will show a complete listing of recognized variables



Libraries / API

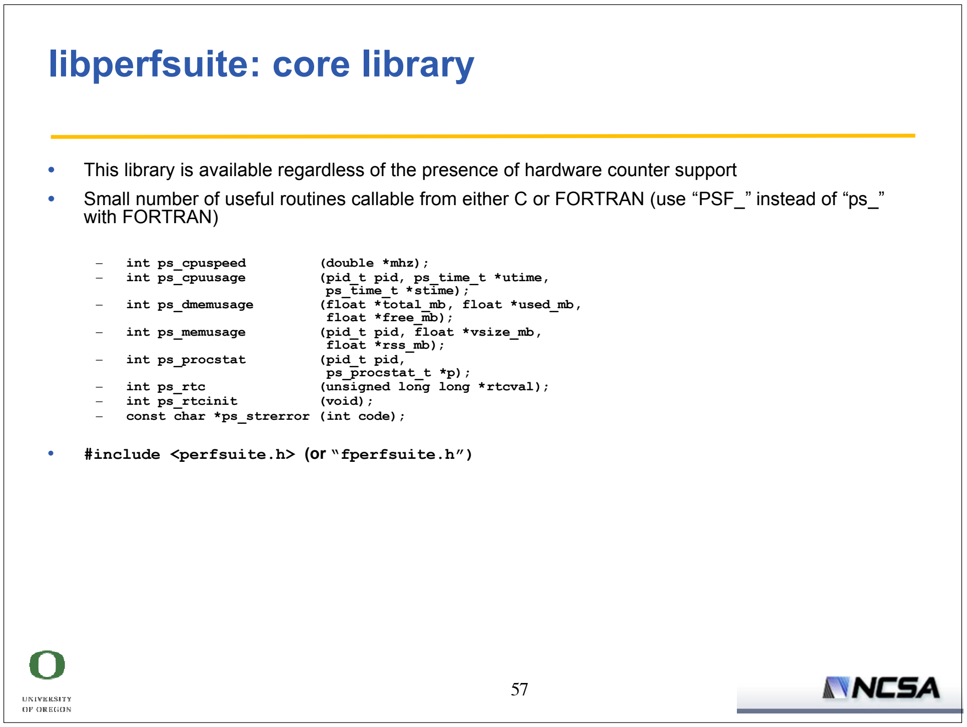
`libperfsuite`

`libpshwpc`



PerfSuite library access (API)

- All of the functionality is also available from within your program (C/C++/Fortran) through a small API
- Same XML documents are read, same XML documents are written, small additional functionality
- Why would you want to use this?
  - Primarily to gain finer control over where measurements are taken in your program. For example, you might defer measurement until program initialization has completed
- For complex uses, you are probably better off using an “industrial-strength” performance library
- The intent of the API is to “abstract out” the process of performance measurement to a very high level

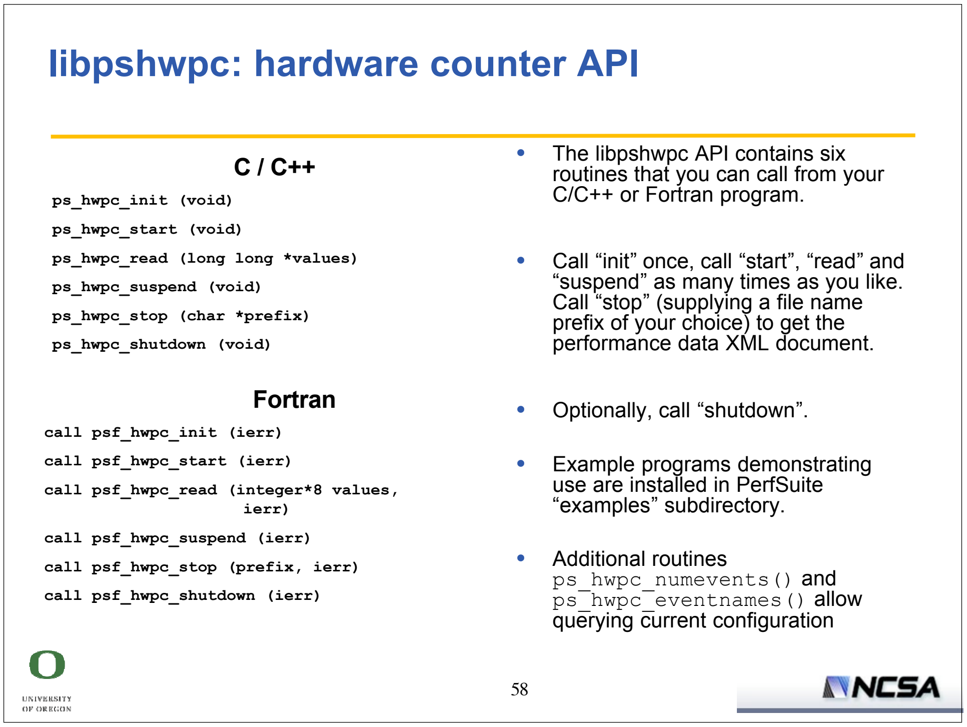


libperfsuite: core library

- This library is available regardless of the presence of hardware counter support
- Small number of useful routines callable from either C or FORTRAN (use “PSF_” instead of “ps_” with FORTRAN)

```
–   int ps_cpuspeed          (double *mhz);
–   int ps_cpuusage          (pid_t pid, ps_time_t *utime,
                              ps_time_t *stime);
–   int ps_dmemusage         (float *total_mb, float *used_mb,
                              float *free_mb);
–   int ps_memusage          (pid_t pid, float *vsize_mb,
                              float *rss_mb);
–   int ps_procstat          (pid_t pid,
                              ps_procstat_t *p);
–   int ps_rtc               (unsigned long long *rtcval);
–   int ps_rtcinit           (void);
–   const char *ps_strerror (int code);
```

- `#include <perfsuite.h>` (or `“fperfsuite.h”`)



libpshwpc: hardware counter API

C / C++

```
ps_hwpc_init (void)
ps_hwpc_start (void)
ps_hwpc_read (long long *values)
ps_hwpc_suspend (void)
ps_hwpc_stop (char *prefix)
ps_hwpc_shutdown (void)
```

Fortran

```
call psf_hwpc_init (ierr)
call psf_hwpc_start (ierr)
call psf_hwpc_read (integer*8 values,
                    ierr)
call psf_hwpc_suspend (ierr)
call psf_hwpc_stop (prefix, ierr)
call psf_hwpc_shutdown (ierr)
```

- The libpshwpc API contains six routines that you can call from your C/C++ or Fortran program.
- Call “init” once, call “start”, “read” and “suspend” as many times as you like. Call “stop” (supplying a file name prefix of your choice) to get the performance data XML document.
- Optionally, call “shutdown”.
- Example programs demonstrating use are installed in PerfSuite “examples” subdirectory.
- Additional routines `ps_hwpc_numevents()` and `ps_hwpc_eventnames()` allow querying current configuration

Example FORTRAN API use

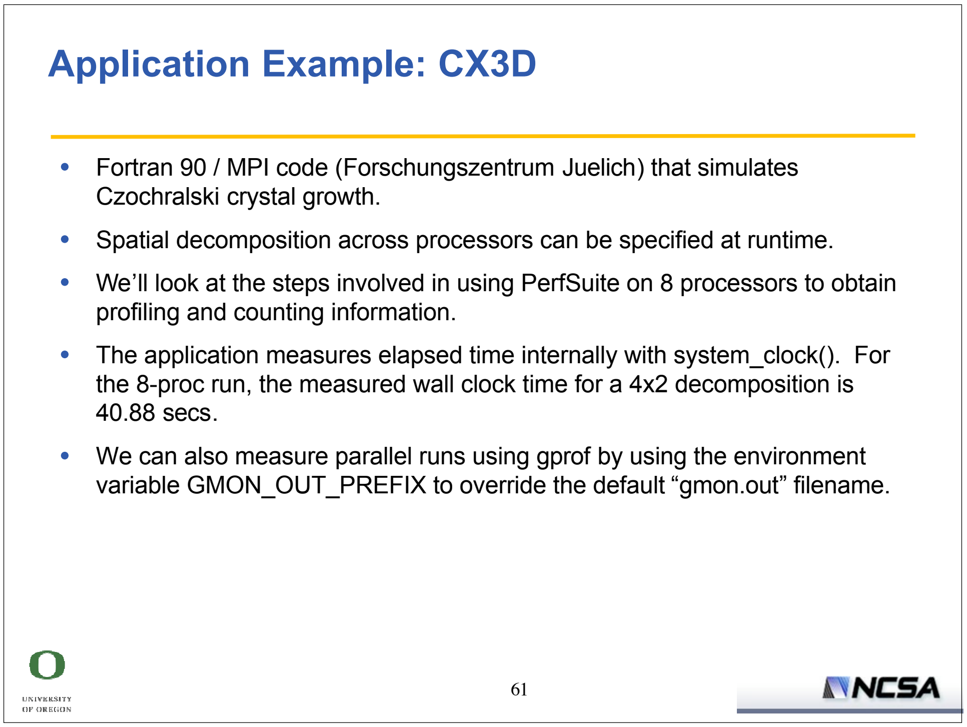
```
include 'fperfsuite.h'
call PSF_hwpc_init(ierr)
call PSF_hwpc_start(ierr)
do j = 1, n
   do i = 1, m
      do k = 1, l
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
      end do
   end do
end do
call PSF_hwpc_stop('perf', ierr)
call PSF_hwpc_shutdown(ierr)
```

```
% ifort -c matmult.f -I/opt/perfsuite/include
% ifort matmult.o -L /usr/apps/tools/perfsuite/lib/intel
    -L/usr/apps/tools/papi/lib -lpshwpc -lperfsuite -lpapi
```

Application Example

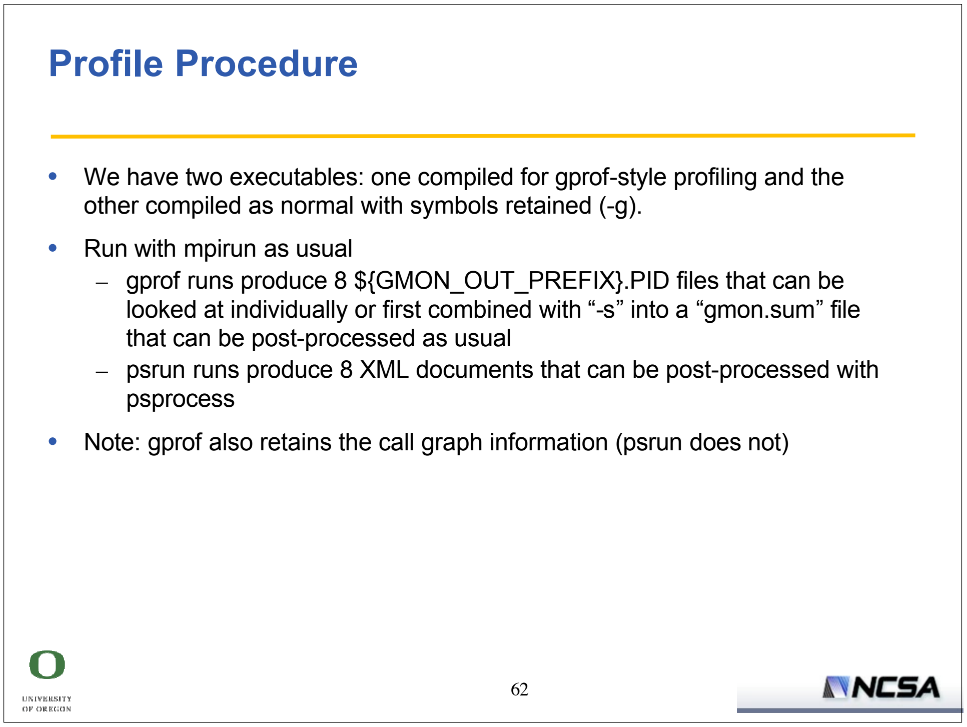
Example acknowledgement:

Felix Wolf, Julich Supercomputing Centre



Application Example: CX3D

- Fortran 90 / MPI code (Forschungszentrum Juelich) that simulates Czochralski crystal growth.
- Spatial decomposition across processors can be specified at runtime.
- We'll look at the steps involved in using PerfSuite on 8 processors to obtain profiling and counting information.
- The application measures elapsed time internally with system_clock(). For the 8-proc run, the measured wall clock time for a 4x2 decomposition is 40.88 secs.
- We can also measure parallel runs using gprof by using the environment variable GMON_OUT_PREFIX to override the default "gmon.out" filename.



Profile Procedure

- We have two executables: one compiled for gprof-style profiling and the other compiled as normal with symbols retained (-g).
- Run with mpirun as usual
  - gprof runs produce 8 ${GMON_OUT_PREFIX}.PID files that can be looked at individually or first combined with "-s" into a "gmon.sum" file that can be post-processed as usual
  - psrun runs produce 8 XML documents that can be post-processed with psprocess
- Note: gprof also retains the call graph information (psrun does not)

Profiling Results (gprof summary)

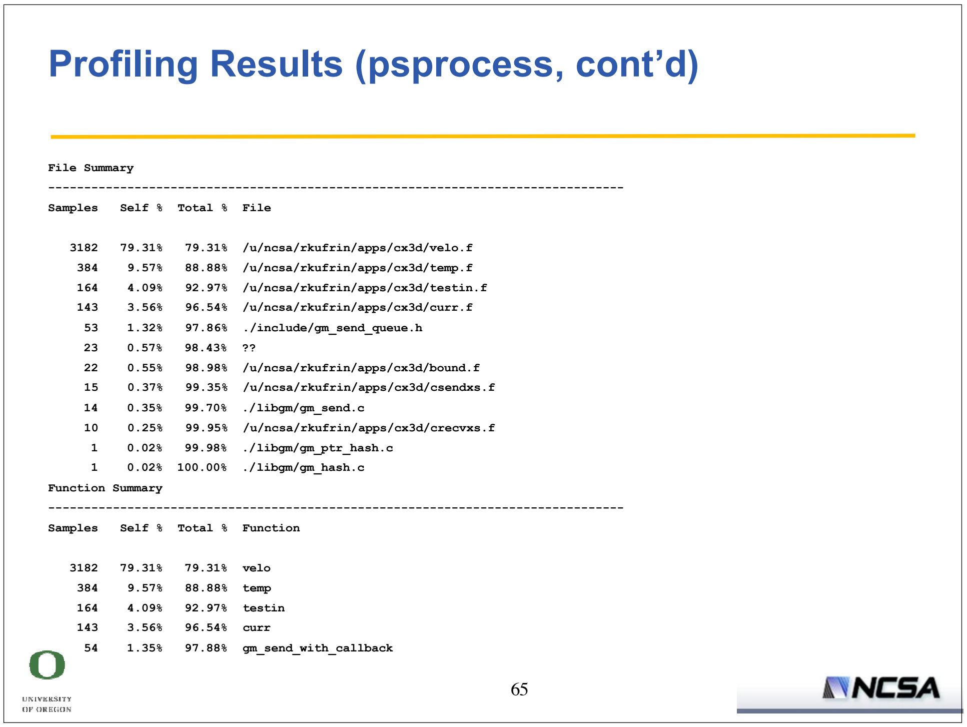
```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 76.79    246.25   246.25     8000    30.78    30.93  velo_
  9.01    275.15    28.90     8000     3.61     3.64  temp_
  3.74    287.14    11.99     8000     1.50     1.50  curr_
  2.04    293.68     6.54                             gmpi_net_lookup
  1.81    299.49     5.81                             gm_ntoh_u8
  1.31    303.69     4.21                            MPID_RecvComplete
  0.75    306.12     2.42                             _gm_ntoh_u8
  0.71    308.38     2.27     8008     0.28     0.32  bound_
```

% time attributed to the highest routine (velo) ranges from 79.21 to 74.42.

```
$ gprof –s cx.gprof ${GMON_OUT_PREFIX}.*
$ gprof –s cx.gprof gmon.sum
```

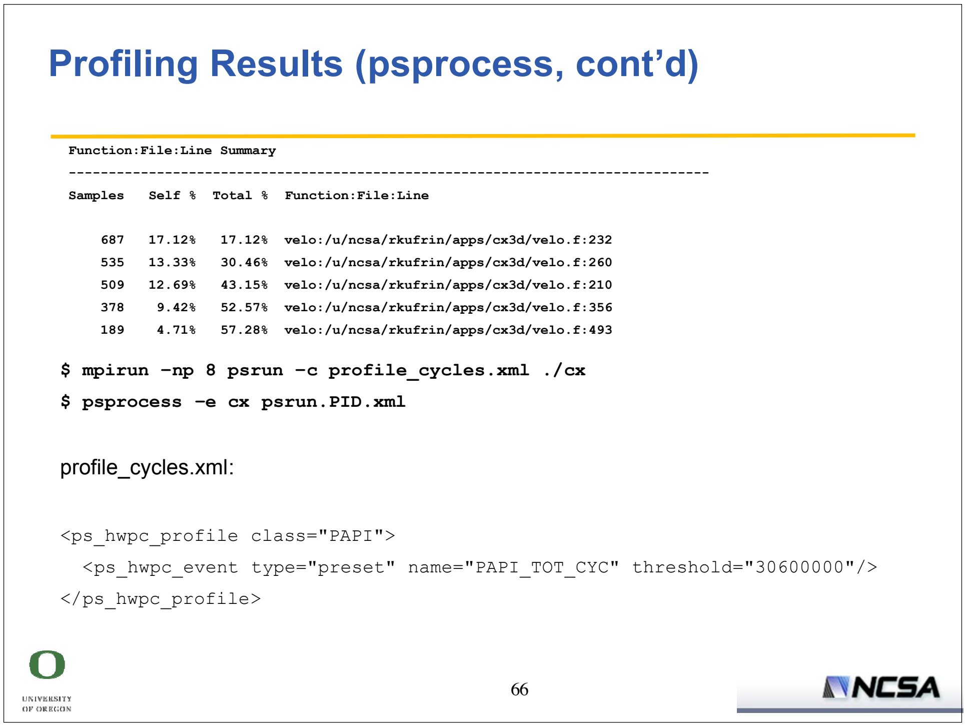
Profiling Results (psprocess individual)

```
Profile Information
========================================================================
Class                         : PAPI
Event                         : PAPI_TOT_CYC (Total cycles)
Period                        : 30600000
Samples                       : 4012
Domain                        : user
Run Time                      : 40.65 (seconds)
Min Self %                    : (all)

Module Summary
------------------------------------------------------------------------
Samples   Self %  Total %  Module

   3942   98.26%   98.26%  /u/ncsa/rkufrin/apps/cx3d/cx
     69    1.72%   99.98%  /opt/gm/lib/libgm.so.0.0.0
      1    0.02%  100.00%  /lib/tls/libpthread-0.34.so
```



Profiling Results (psprocess, cont’d)

```
File Summary
--------------------------------------------------------------------------------
Samples   Self %   Total %   File

   3182   79.31%    79.31%   /u/ncsa/rkufrin/apps/cx3d/velo.f
    384    9.57%    88.88%   /u/ncsa/rkufrin/apps/cx3d/temp.f
    164    4.09%    92.97%   /u/ncsa/rkufrin/apps/cx3d/testin.f
    143    3.56%    96.54%   /u/ncsa/rkufrin/apps/cx3d/curr.f
     53    1.32%    97.86%   ./include/gm_send_queue.h
     23    0.57%    98.43%   ??
     22    0.55%    98.98%   /u/ncsa/rkufrin/apps/cx3d/bound.f
     15    0.37%    99.35%   /u/ncsa/rkufrin/apps/cx3d/csendxs.f
     14    0.35%    99.70%   ./libgm/gm_send.c
     10    0.25%    99.95%   /u/ncsa/rkufrin/apps/cx3d/crecvxs.f
      1    0.02%    99.98%   ./libgm/gm_ptr_hash.c
      1    0.02%   100.00%   ./libgm/gm_hash.c
Function Summary
--------------------------------------------------------------------------------
Samples   Self %   Total %   Function

   3182   79.31%    79.31%   velo
    384    9.57%    88.88%   temp
    164    4.09%    92.97%   testin
    143    3.56%    96.54%   curr
     54    1.35%    97.88%   gm_send_with_callback
```



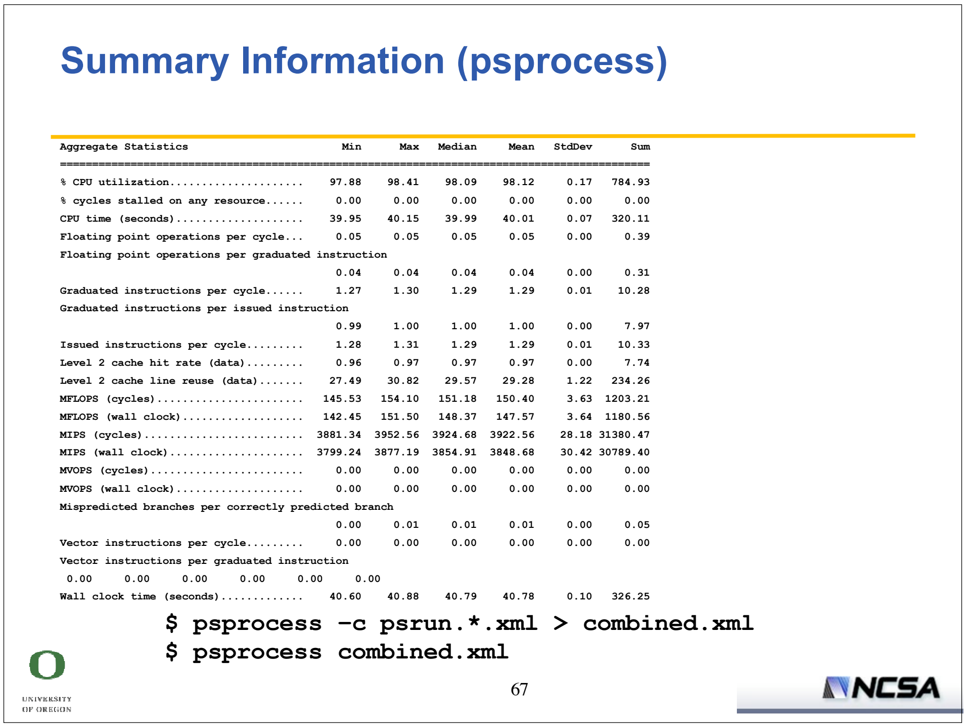
Profiling Results (psprocess, cont’d)

```
Function:File:Line Summary
--------------------------------------------------------------------------------
Samples   Self %   Total %   Function:File:Line

    687   17.12%    17.12%   velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:232
    535   13.33%    30.46%   velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:260
    509   12.69%    43.15%   velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:210
    378    9.42%    52.57%   velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:356
    189    4.71%    57.28%   velo:/u/ncsa/rkufrin/apps/cx3d/velo.f:493
```

```
$ mpirun -np 8 psrun -c profile_cycles.xml ./cx
$ psprocess -e cx psrun.PID.xml
```

profile_cycles.xml:

```
<ps_hwpc_profile class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_TOT_CYC" threshold="30600000"/>
</ps_hwpc_profile>
```

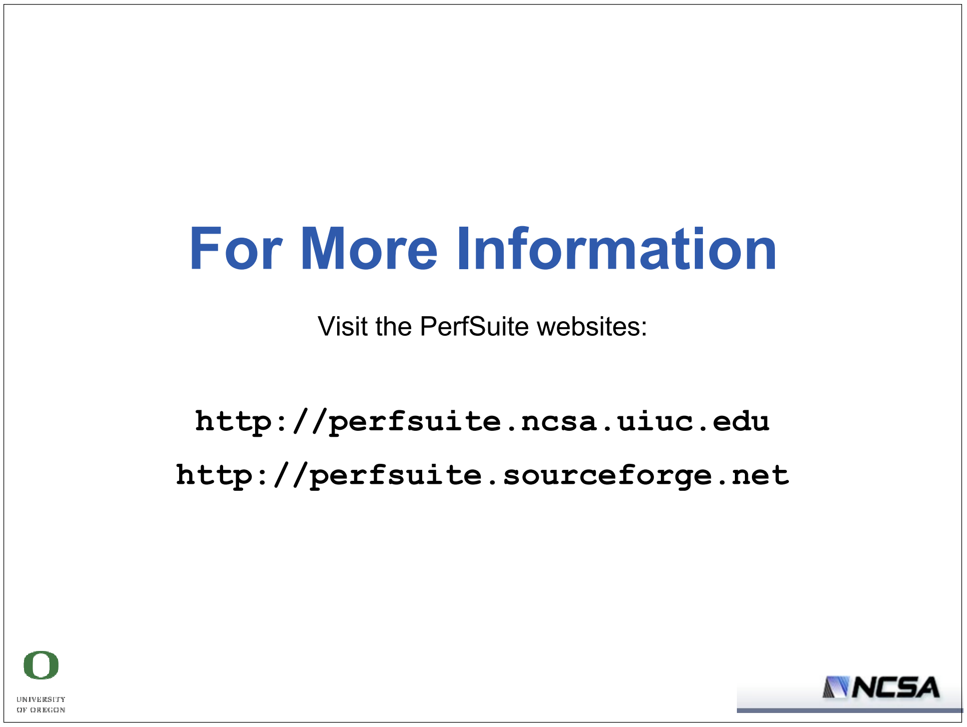


Summary Information (psprocess)

```
Aggregate Statistics                        Min      Max   Median     Mean   StdDev      Sum
============================================================================================
% CPU utilization.....................    97.88    98.41    98.09    98.12     0.17   784.93
% cycles stalled on any resource......     0.00     0.00     0.00     0.00     0.00     0.00
CPU time (seconds)....................    39.95    40.15    39.99    40.01     0.07   320.11
Floating point operations per cycle...     0.05     0.05     0.05     0.05     0.00     0.39
Floating point operations per graduated instruction
                                           0.04     0.04     0.04     0.04     0.00     0.31
Graduated instructions per cycle......     1.27     1.30     1.29     1.29     0.01    10.28
Graduated instructions per issued instruction
                                           0.99     1.00     1.00     1.00     0.00     7.97
Issued instructions per cycle.........     1.28     1.31     1.29     1.29     0.01    10.33
Level 2 cache hit rate (data).........     0.96     0.97     0.97     0.97     0.00     7.74
Level 2 cache line reuse (data).......    27.49    30.82    29.57    29.28     1.22   234.26
MFLOPS (cycles).......................   145.53   154.10   151.18   150.40     3.63  1203.21
MFLOPS (wall clock)...................   142.45   151.50   148.37   147.57     3.64  1180.56
MIPS (cycles).........................  3881.34  3952.56  3924.68  3922.56    28.18 31380.47
MIPS (wall clock).....................  3799.24  3877.19  3854.91  3848.68    30.42 30789.40
MVOPS (cycles)........................     0.00     0.00     0.00     0.00     0.00     0.00
MVOPS (wall clock)....................     0.00     0.00     0.00     0.00     0.00     0.00
Mispredicted branches per correctly predicted branch
                                           0.00     0.01     0.01     0.01     0.00     0.05
Vector instructions per cycle.........     0.00     0.00     0.00     0.00     0.00     0.00
Vector instructions per graduated instruction
 0.00     0.00     0.00     0.00     0.00     0.00
Wall clock time (seconds).............    40.60    40.88    40.79    40.78     0.10   326.25
```

```
$ psprocess –c psrun.*.xml > combined.xml
$ psprocess combined.xml
```

67

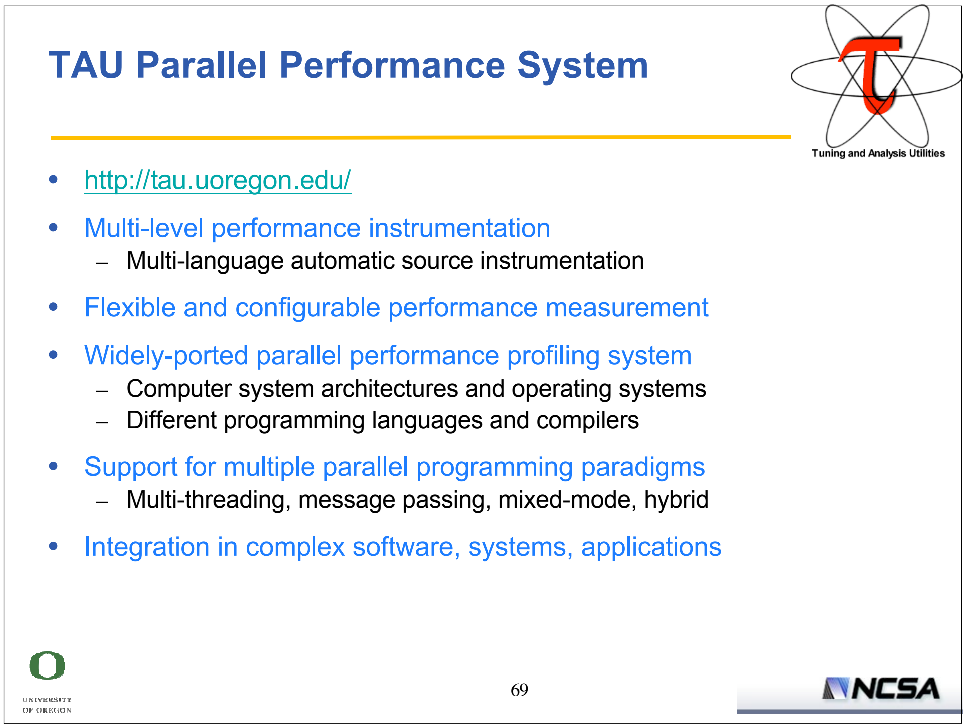


For More Information

Visit the PerfSuite websites:

http://perfsuite.ncsa.uiuc.edu

http://perfsuite.sourceforge.net



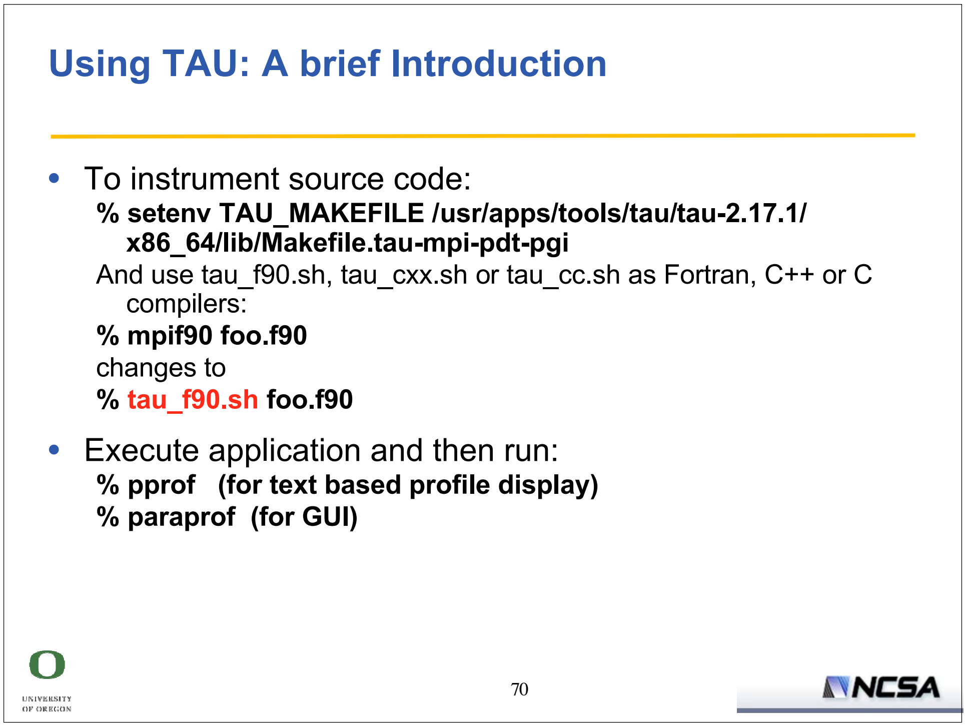
TAU Parallel Performance System

Tuning and Analysis Utilities

- 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

UNIVERSITY OF OREGON

NCSA



Using TAU: A brief Introduction

- To instrument source code:

  % setenv TAU_MAKEFILE /usr/apps/tools/tau/tau-2.17.1/x86_64/lib/Makefile.tau-mpi-pdt-pgi

  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

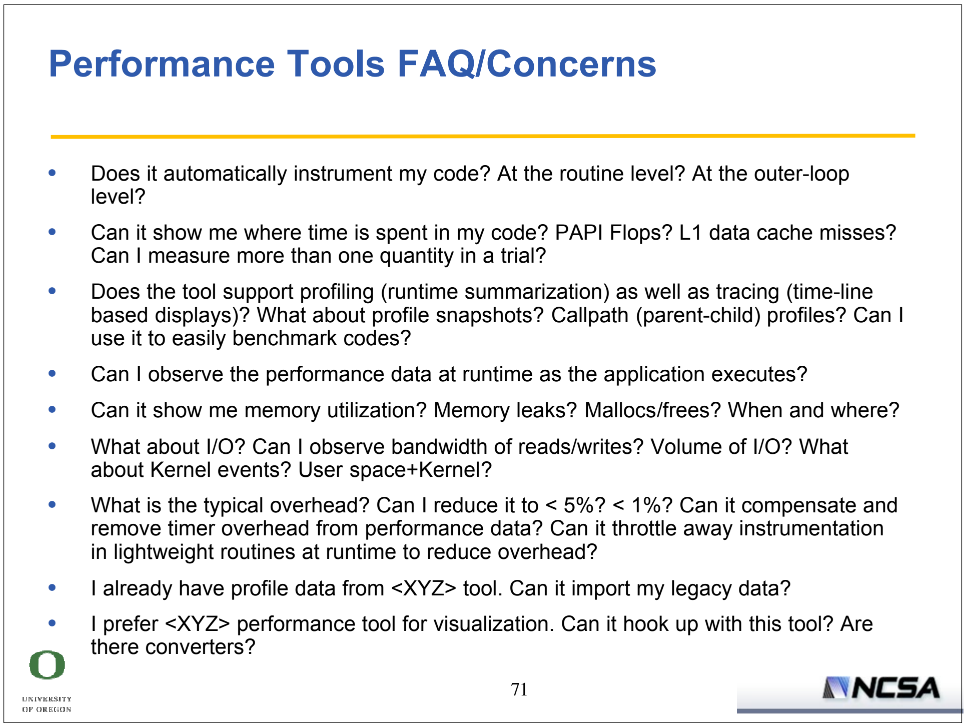
- Execute application and then run:

  % pprof   (for text based profile display)

  % paraprof (for GUI)

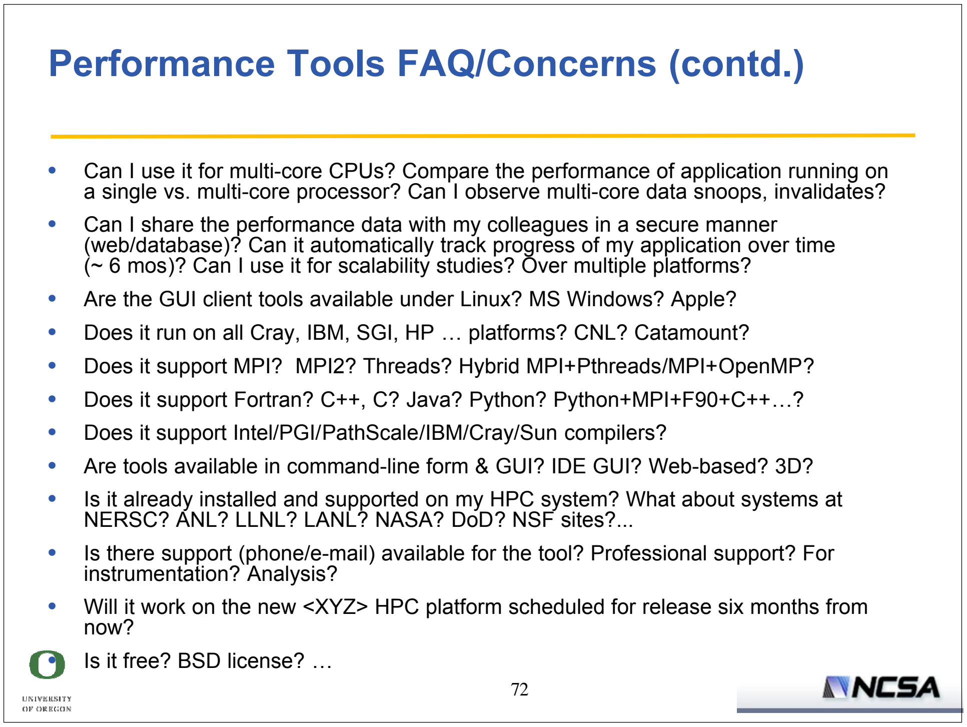
UNIVERSITY OF OREGON

NCSA



Performance Tools FAQ/Concerns

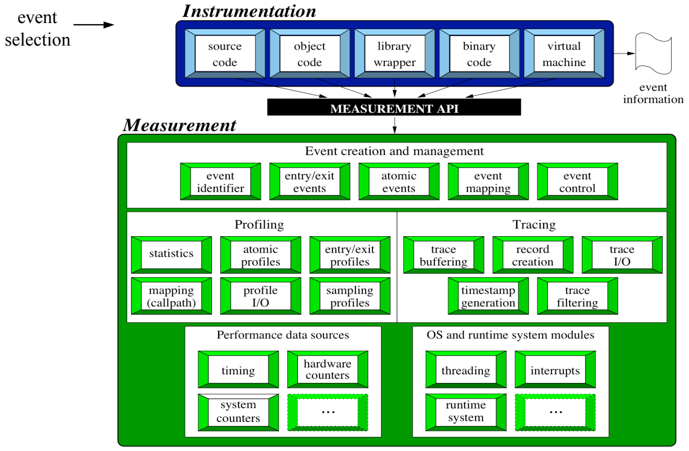
- Does it automatically instrument my code? At the routine level? At the outer-loop level?
- Can it show me where time is spent in my code? PAPI Flops? L1 data cache misses? Can I measure more than one quantity in a trial?
- Does the tool support profiling (runtime summarization) as well as tracing (time-line based displays)? What about profile snapshots? Callpath (parent-child) profiles? Can I use it to easily benchmark codes?
- Can I observe the performance data at runtime as the application executes?
- Can it show me memory utilization? Memory leaks? Mallocs/frees? When and where?
- What about I/O? Can I observe bandwidth of reads/writes? Volume of I/O? What about Kernel events? User space+Kernel?
- What is the typical overhead? Can I reduce it to < 5%? < 1%? Can it compensate and remove timer overhead from performance data? Can it throttle away instrumentation in lightweight routines at runtime to reduce overhead?
- I already have profile data from <XYZ> tool. Can it import my legacy data?
- I prefer <XYZ> performance tool for visualization. Can it hook up with this tool? Are there converters?



Performance Tools FAQ/Concerns (contd.)

- Can I use it for multi-core CPUs? Compare the performance of application running on a single vs. multi-core processor? Can I observe multi-core data snoops, invalidates?
- Can I share the performance data with my colleagues in a secure manner (web/database)? Can it automatically track progress of my application over time (~ 6 mos)? Can I use it for scalability studies? Over multiple platforms?
- Are the GUI client tools available under Linux? MS Windows? Apple?
- Does it run on all Cray, IBM, SGI, HP … platforms? CNL? Catamount?
- Does it support MPI? MPI2? Threads? Hybrid MPI+Pthreads/MPI+OpenMP?
- Does it support Fortran? C++, C? Java? Python? Python+MPI+F90+C++…?
- Does it support Intel/PGI/PathScale/IBM/Cray/Sun compilers?
- Are tools available in command-line form & GUI? IDE GUI? Web-based? 3D?
- Is it already installed and supported on my HPC system? What about systems at NERSC? ANL? LLNL? LANL? NASA? DoD? NSF sites?...
- Is there support (phone/e-mail) available for the tool? Professional support? For instrumentation? Analysis?
- Will it work on the new <XYZ> HPC platform scheduled for release six months from now?
- Is it free? BSD license? …

TAU Performance System Architecture



event selection

Instrumentation

source code

object code

library wrapper

binary code

virtual machine

event information

MEASUREMENT API

Measurement

Event creation and management

event identifier

entry/exit events

atomic events

event mapping

event control

Profiling

statistics

atomic profiles

entry/exit profiles

mapping (callpath)

profile I/O

sampling profiles

Tracing

trace buffering

record creation

trace I/O

timestamp generation

trace filtering

Performance data sources

timing

hardware counters

system counters

. . .

OS and runtime system modules

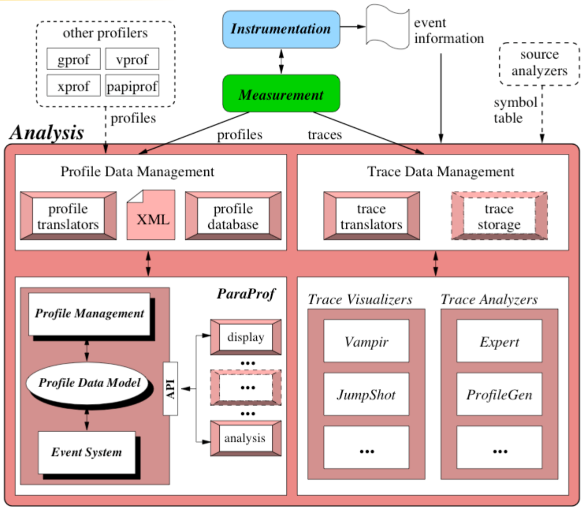
threading

interrupts

runtime system

. . .

TAU Performance System Architecture



other profilers

gprof

vprof

xprof

papiprof

profiles

Instrumentation

event information

Measurement

source analyzers

symbol table

Analysis

profiles

traces

Profile Data Management

profile translators

XML

profile database

Trace Data Management

trace translators

trace storage

ParaProf

Profile Management

Profile Data Model

Event System

API

display

•••

•••

•••

analysis

Trace Visualizers

Vampir

JumpShot

•••

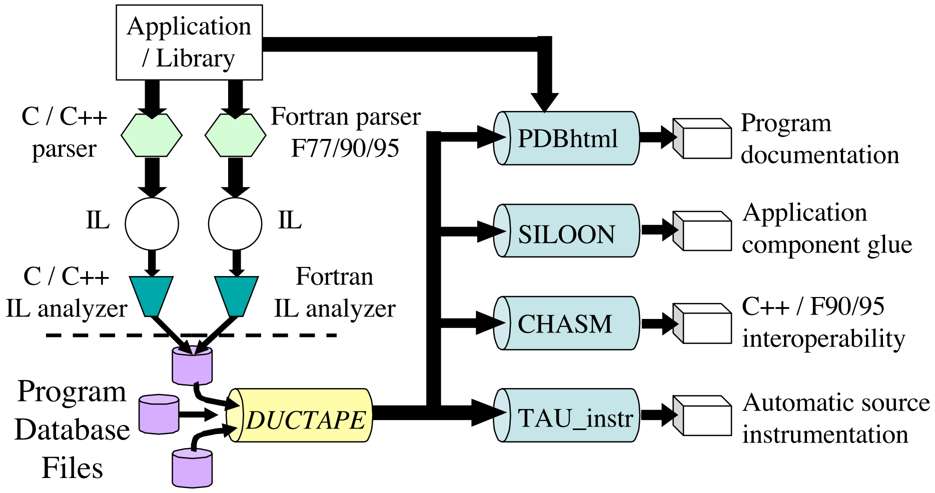
Trace Analyzers

Expert

ProfileGen

•••

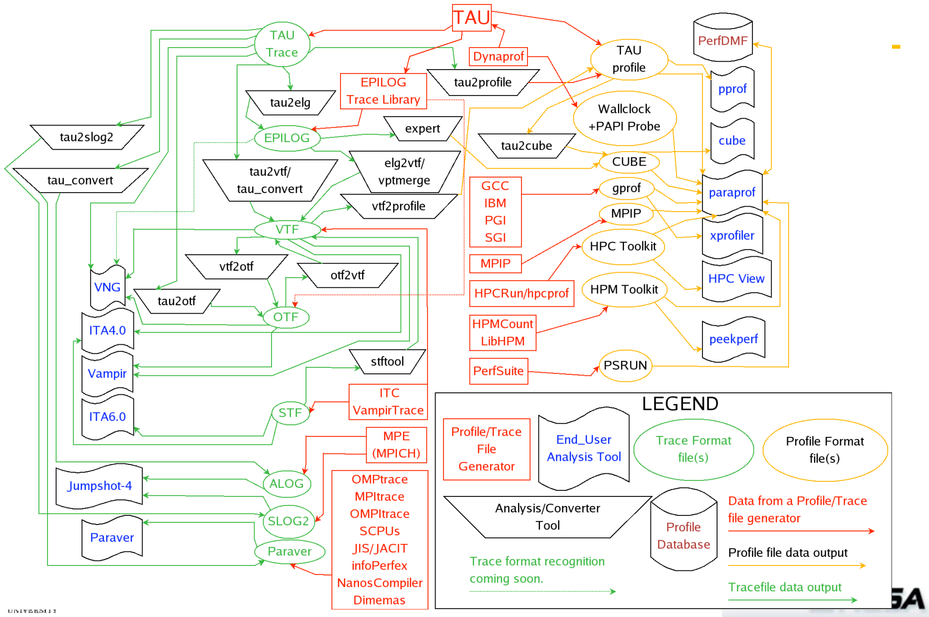
Program Database Toolkit (PDT)



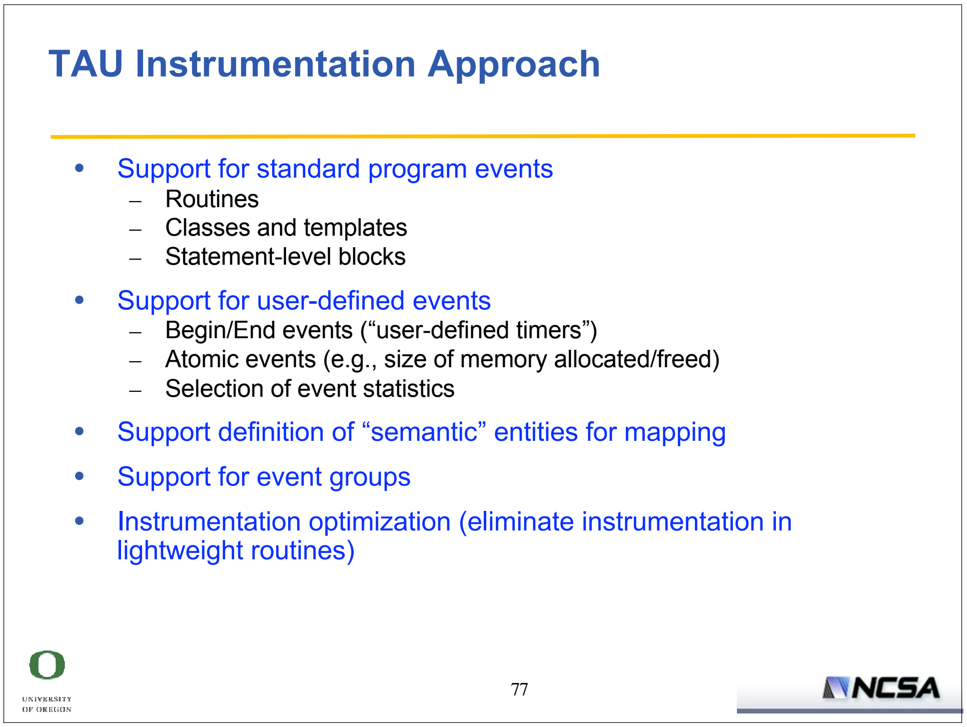
UNIVERSITY OF OREGON

NCSA

Building Bridges to Other Tools

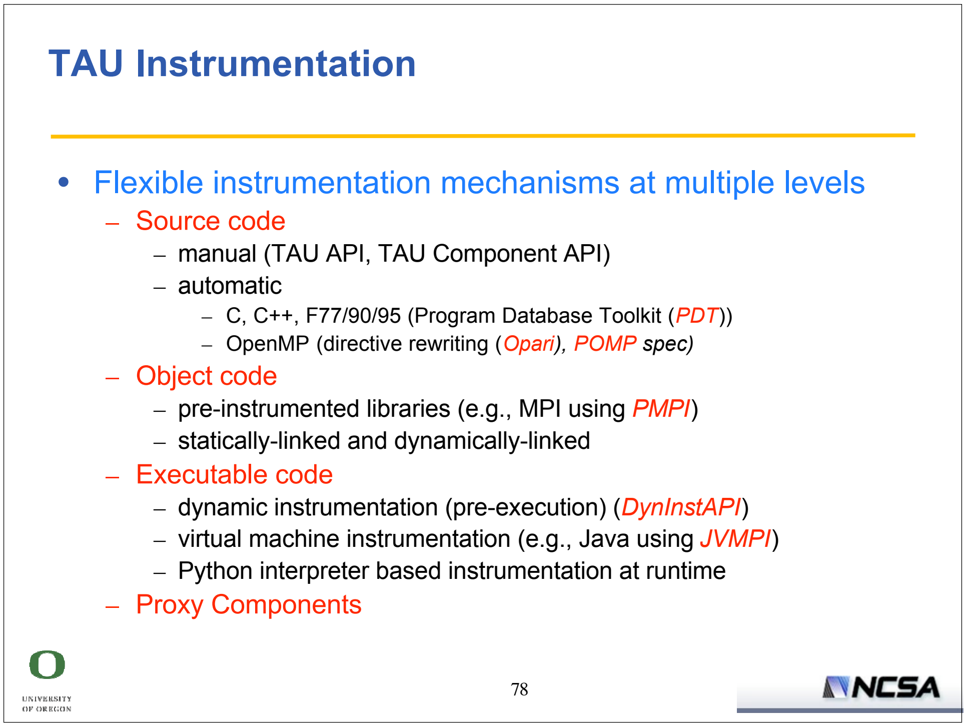


UNIVERSITY OF OREGON



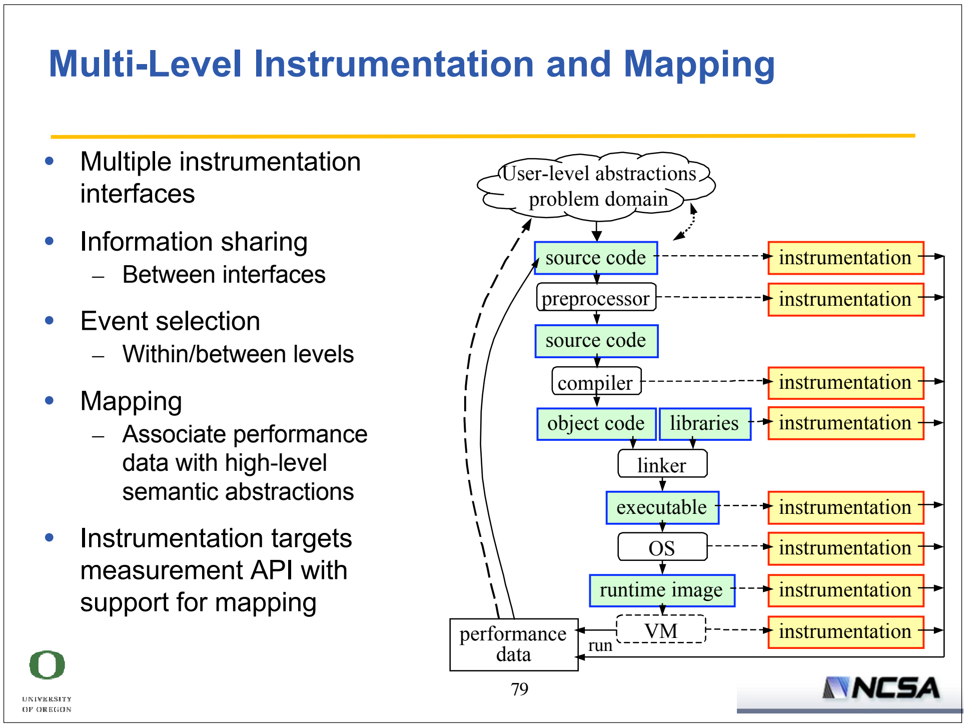
TAU Instrumentation Approach

- Support for standard program events
  - Routines
  - Classes and templates
  - Statement-level blocks
- Support for user-defined events
  - Begin/End events (“user-defined timers”)
  - Atomic events (e.g., size of memory allocated/freed)
  - Selection of event statistics
- Support definition of “semantic” entities for mapping
- Support for event groups
- Instrumentation optimization (eliminate instrumentation in lightweight routines)



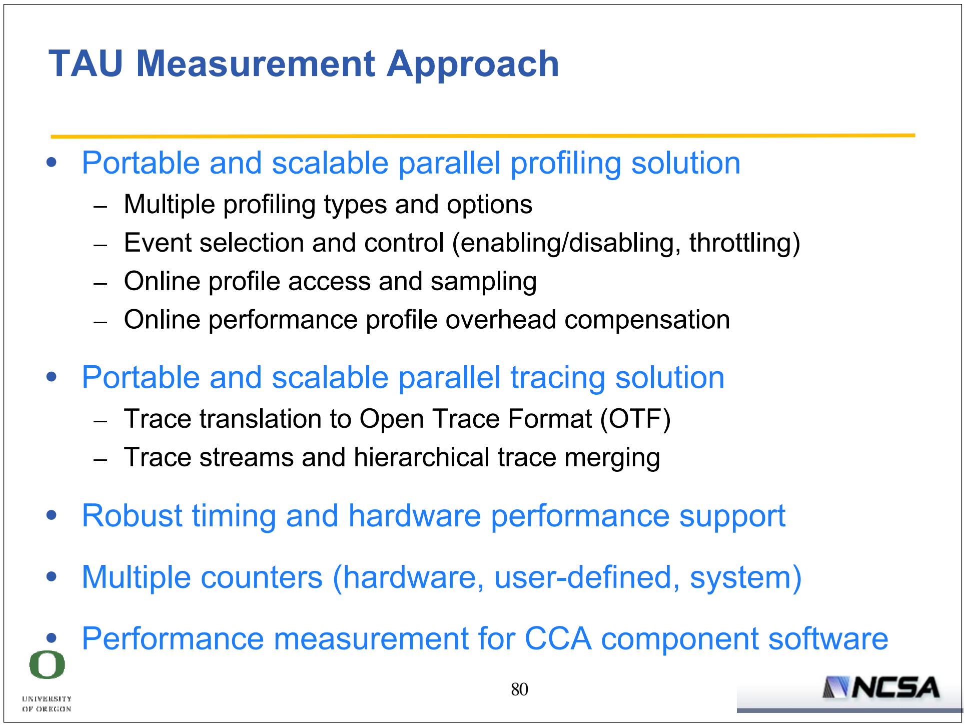
TAU Instrumentation

- Flexible instrumentation mechanisms at multiple levels
  - Source code
    - manual (TAU API, TAU Component API)
    - automatic
      - C, C++, F77/90/95 (Program Database Toolkit (*PDT*))
      - OpenMP (directive rewriting (*Opari*), *POMP spec)*
  - Object code
    - pre-instrumented libraries (e.g., MPI using *PMPI*)
    - statically-linked and dynamically-linked
  - Executable code
    - dynamic instrumentation (pre-execution) (*DynInstAPI*)
    - virtual machine instrumentation (e.g., Java using *JVMPI*)
    - Python interpreter based instrumentation at runtime
  - Proxy Components



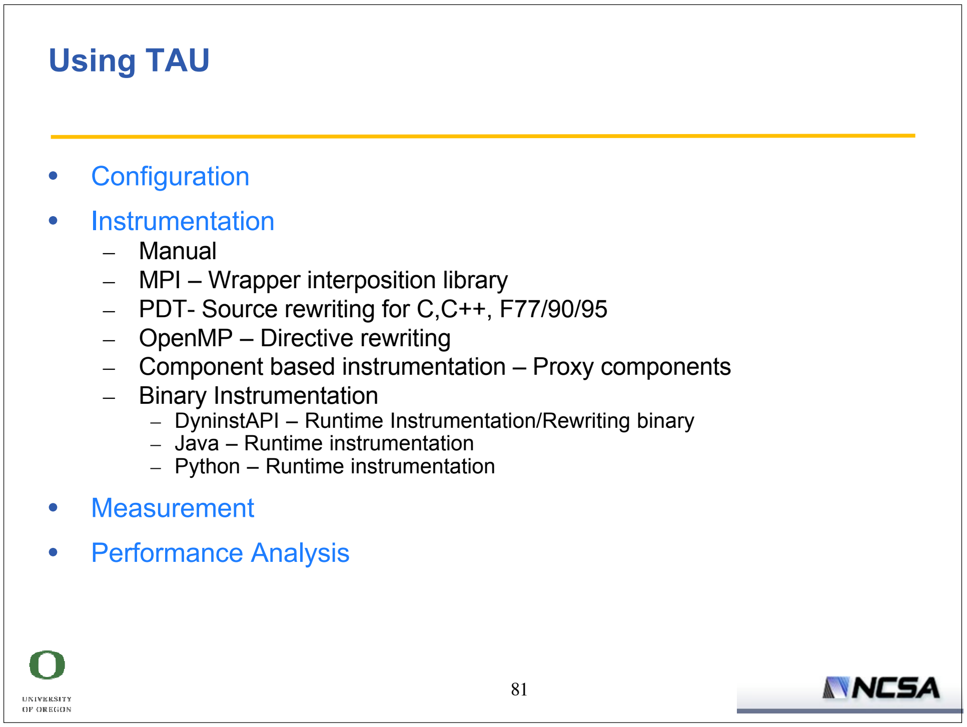
Multi-Level Instrumentation and Mapping

- Multiple instrumentation interfaces
- Information sharing
  - Between interfaces
- Event selection
  - Within/between levels
- Mapping
  - Associate performance data with high-level semantic abstractions
- Instrumentation targets measurement API with support for mapping



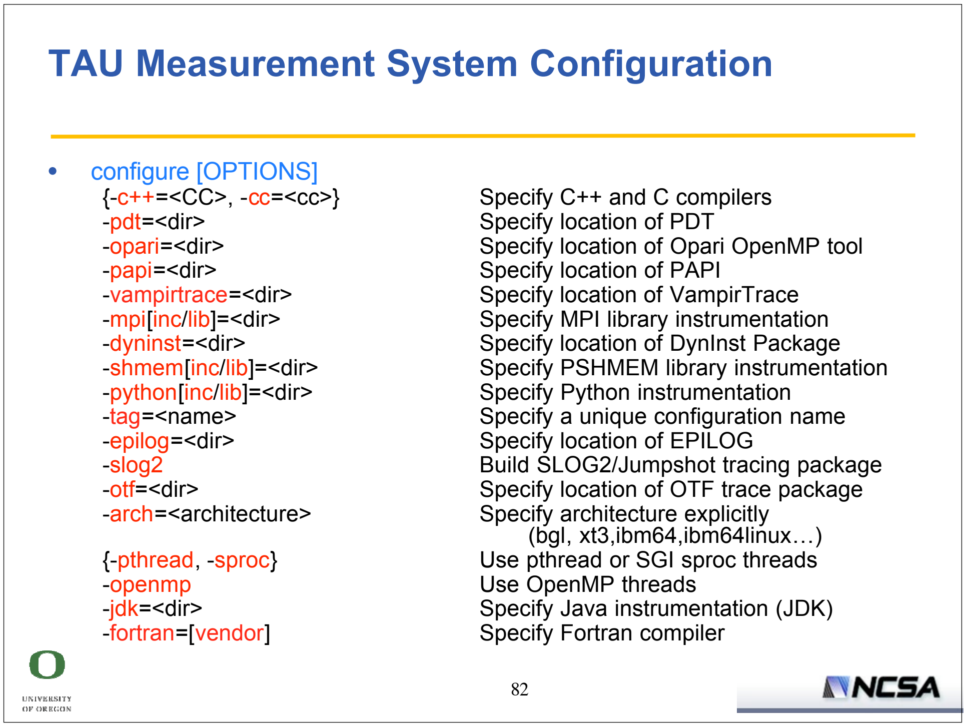
TAU Measurement Approach

- Portable and scalable parallel profiling solution
  - Multiple profiling types and options
  - Event selection and control (enabling/disabling, throttling)
  - Online profile access and sampling
  - Online performance profile overhead compensation
- Portable and scalable parallel tracing solution
  - Trace translation to Open Trace Format (OTF)
  - Trace streams and hierarchical trace merging
- Robust timing and hardware performance support
- Multiple counters (hardware, user-defined, system)
- Performance measurement for CCA component software



Using TAU

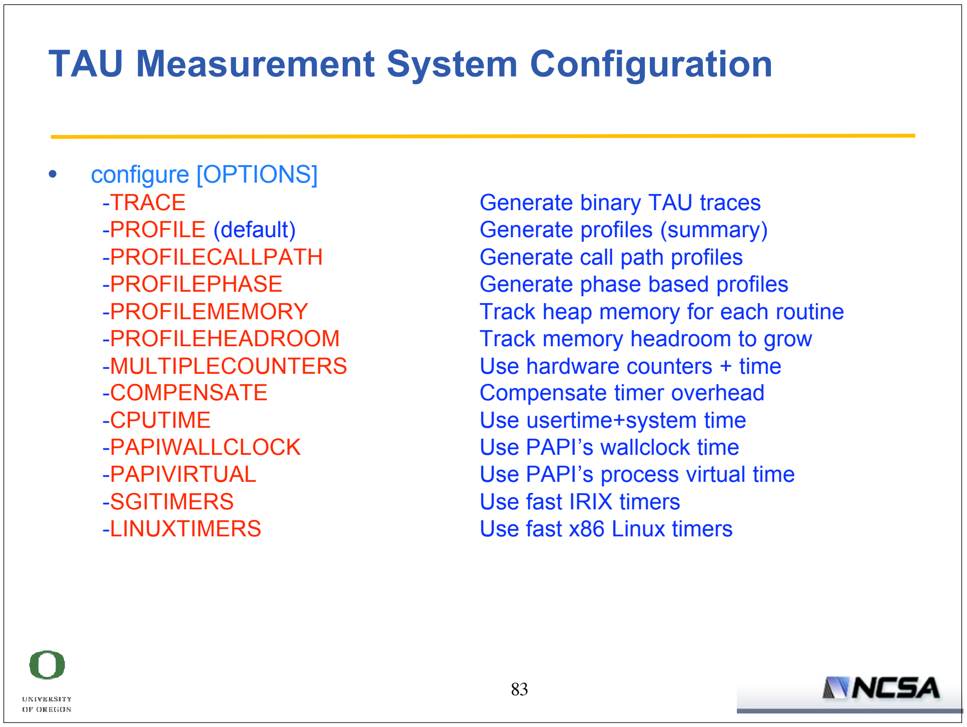
- Configuration
- Instrumentation
  - Manual
  - MPI – Wrapper interposition library
  - PDT- Source rewriting for C,C++, F77/90/95
  - OpenMP – Directive rewriting
  - Component based instrumentation – Proxy components
  - Binary Instrumentation
    - DyninstAPI – Runtime Instrumentation/Rewriting binary
    - Java – Runtime instrumentation
    - Python – Runtime instrumentation
- Measurement
- Performance Analysis



TAU Measurement System Configuration

- configure [OPTIONS]

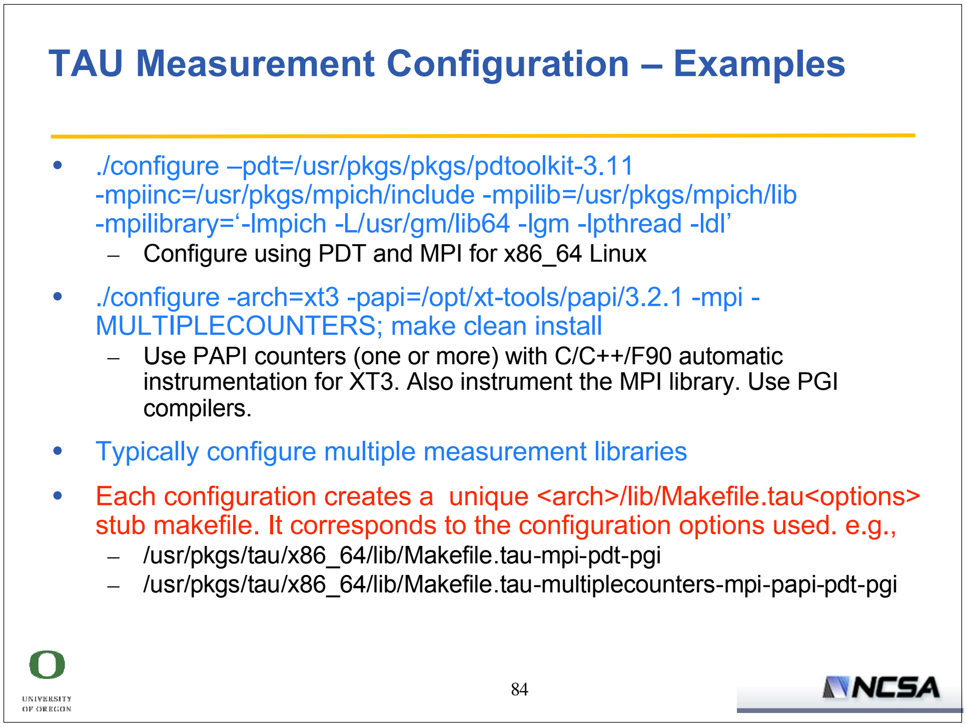
{-c++=<CC>, -cc=<cc>}	Specify C++ and C compilers
-pdt=<dir>	Specify location of PDT
-opari=<dir>	Specify location of Opari OpenMP tool
-papi=<dir>	Specify location of PAPI
-vampirtrace=<dir>	Specify location of VampirTrace
-mpi[inc/lib]=<dir>	Specify MPI library instrumentation
-dyninst=<dir>	Specify location of DynInst Package
-shmem[inc/lib]=<dir>	Specify PSHMEM library instrumentation
-python[inc/lib]=<dir>	Specify Python instrumentation
-tag=<name>	Specify a unique configuration name
-epilog=<dir>	Specify location of EPILOG
-slog2	Build SLOG2/Jumpshot tracing package
-otf=<dir>	Specify location of OTF trace package
-arch=<architecture>	Specify architecture explicitly (bgl, xt3,ibm64,ibm64linux…)
{-pthread, -sproc}	Use pthread or SGI sproc threads
-openmp	Use OpenMP threads
-jdk=<dir>	Specify Java instrumentation (JDK)
-fortran=[vendor]	Specify Fortran compiler



TAU Measurement System Configuration

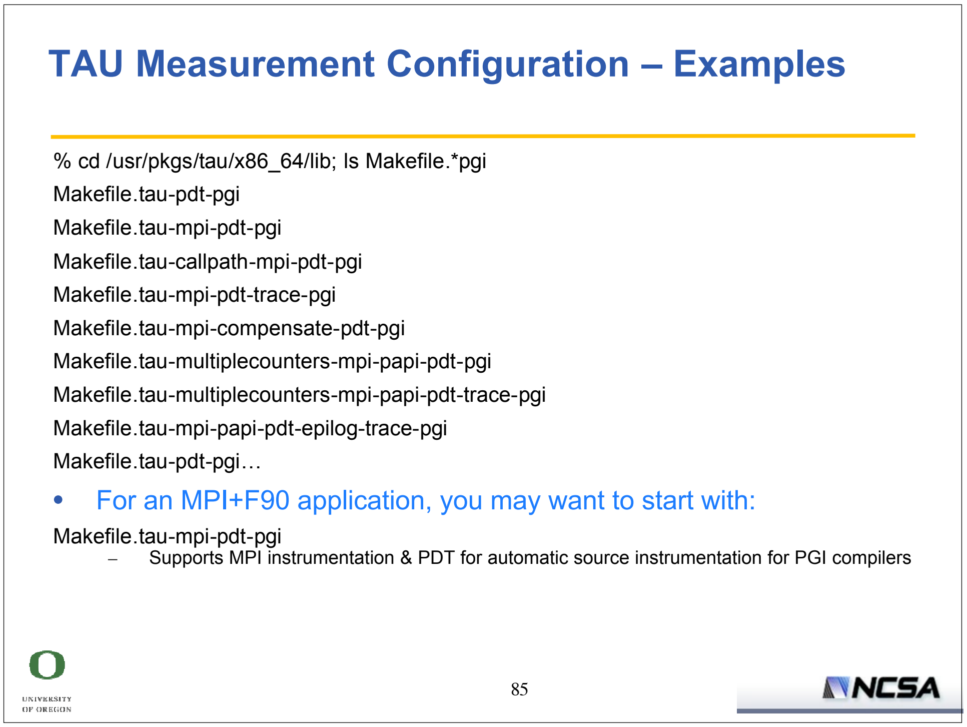
- configure [OPTIONS]

-TRACE	Generate binary TAU traces
-PROFILE (default)	Generate profiles (summary)
-PROFILECALLPATH	Generate call path profiles
-PROFILEPHASE	Generate phase based profiles
-PROFILEMEMORY	Track heap memory for each routine
-PROFILEHEADROOM	Track memory headroom to grow
-MULTIPLECOUNTERS	Use hardware counters + time
-COMPENSATE	Compensate timer overhead
-CPUTIME	Use usertime+system time
-PAPIWALLCLOCK	Use PAPI’s wallclock time
-PAPIVIRTUAL	Use PAPI’s process virtual time
-SGITIMERS	Use fast IRIX timers
-LINUXTIMERS	Use fast x86 Linux timers



TAU Measurement Configuration – Examples

- ./configure –pdt=/usr/pkgs/pkgs/pdtoolkit-3.11 -mpiinc=/usr/pkgs/mpich/include -mpilib=/usr/pkgs/mpich/lib -mpilibrary=‘-lmpich -L/usr/gm/lib64 -lgm -lpthread -ldl’
  - Configure using PDT and MPI for x86_64 Linux
- ./configure -arch=xt3 -papi=/opt/xt-tools/papi/3.2.1 -mpi -MULTIPLECOUNTERS; make clean install
  - Use PAPI counters (one or more) with C/C++/F90 automatic instrumentation for XT3. Also instrument the MPI library. Use PGI compilers.
- Typically configure multiple measurement libraries
- Each configuration creates a unique <arch>/lib/Makefile.tau<options> stub makefile. It corresponds to the configuration options used. e.g.,
  - /usr/pkgs/tau/x86_64/lib/Makefile.tau-mpi-pdt-pgi
  - /usr/pkgs/tau/x86_64/lib/Makefile.tau-multiplecounters-mpi-papi-pdt-pgi



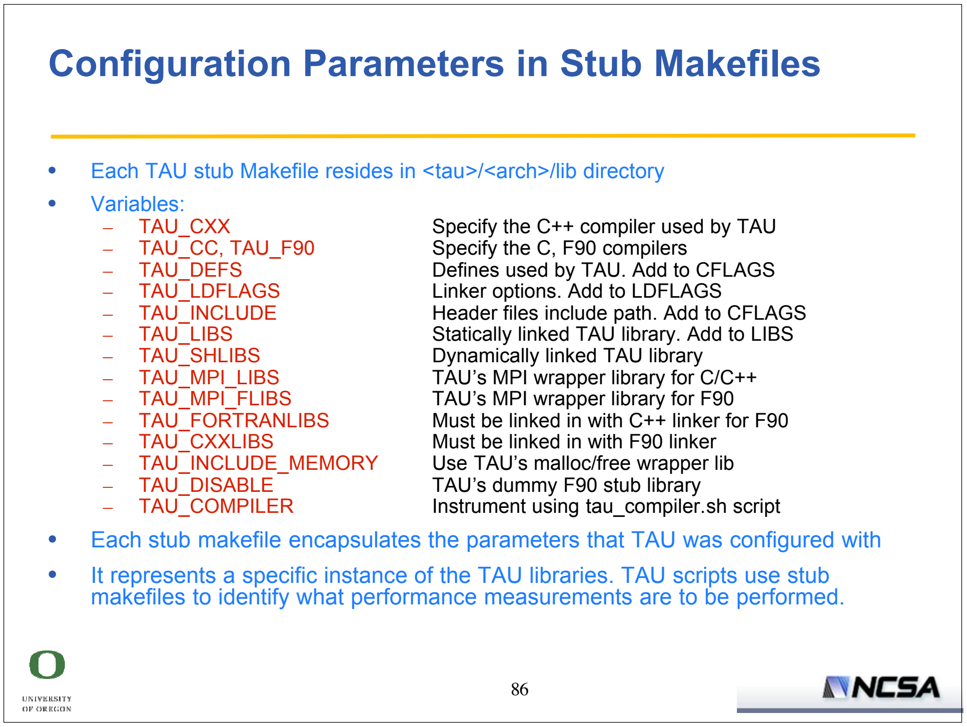
TAU Measurement Configuration – Examples

```
% cd /usr/pkgs/tau/x86_64/lib; ls Makefile.*pgi
Makefile.tau-pdt-pgi
Makefile.tau-mpi-pdt-pgi
Makefile.tau-callpath-mpi-pdt-pgi
Makefile.tau-mpi-pdt-trace-pgi
Makefile.tau-mpi-compensate-pdt-pgi
Makefile.tau-multiplecounters-mpi-papi-pdt-pgi
Makefile.tau-multiplecounters-mpi-papi-pdt-trace-pgi
Makefile.tau-mpi-papi-pdt-epilog-trace-pgi
Makefile.tau-pdt-pgi…
```

- For an MPI+F90 application, you may want to start with:

Makefile.tau-mpi-pdt-pgi

- – Supports MPI instrumentation & PDT for automatic source instrumentation for PGI compilers



Configuration Parameters in Stub Makefiles

- Each TAU stub Makefile resides in <tau>/<arch>/lib directory
- Variables:

– TAU_CXX	Specify the C++ compiler used by TAU
– TAU_CC, TAU_F90	Specify the C, F90 compilers
– TAU_DEFS	Defines used by TAU. Add to CFLAGS
– TAU_LDFLAGS	Linker options. Add to LDFLAGS
– TAU_INCLUDE	Header files include path. Add to CFLAGS
– TAU_LIBS	Statically linked TAU library. Add to LIBS
– TAU_SHLIBS	Dynamically linked TAU library
– TAU_MPI_LIBS	TAU's MPI wrapper library for C/C++
– TAU_MPI_FLIBS	TAU's MPI wrapper library for F90
– TAU_FORTRANLIBS	Must be linked in with C++ linker for F90
– TAU_CXXLIBS	Must be linked in with F90 linker
– TAU_INCLUDE_MEMORY	Use TAU's malloc/free wrapper lib
– TAU_DISABLE	TAU's dummy F90 stub library
– TAU_COMPILER	Instrument using tau_compiler.sh script

- Each stub makefile encapsulates the parameters that TAU was configured with
- It represents a specific instance of the TAU libraries. TAU scripts use stub makefiles to identify what performance measurements are to be performed.

Using TAU

- Install TAU

  % configure [options]; make clean install

- Typically modify application makefile and choose TAU configuration
  - Select TAU's stub makefile, change name of compiler in Makefile

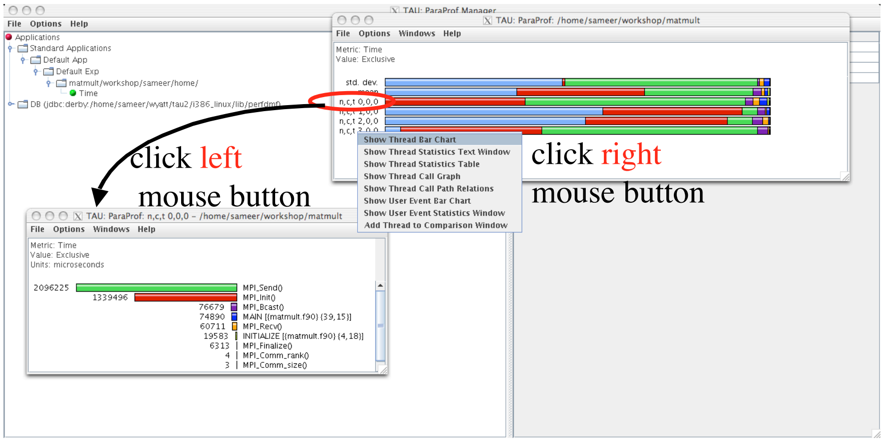
  % setenv TAU_MAKEFILE /usr/pkgs/tau/x86_64/lib/Makefile.tau-mpi-pdt-pgi

  % setenv TAU_OPTIONS '-optVerbose -optKeepFiles ...'
  - F90 = tau_f90.sh CXX = tau_cxx.sh CC = tau_cc.sh
- Set environment variables
  - Directory where profiles/traces are to be stored/counter selection
- Execute application

  % mpirun –np <procs> a.out;

- Analyze performance data
  - paraprof, vampir, pprof, paraver …

ParaProf Main Window

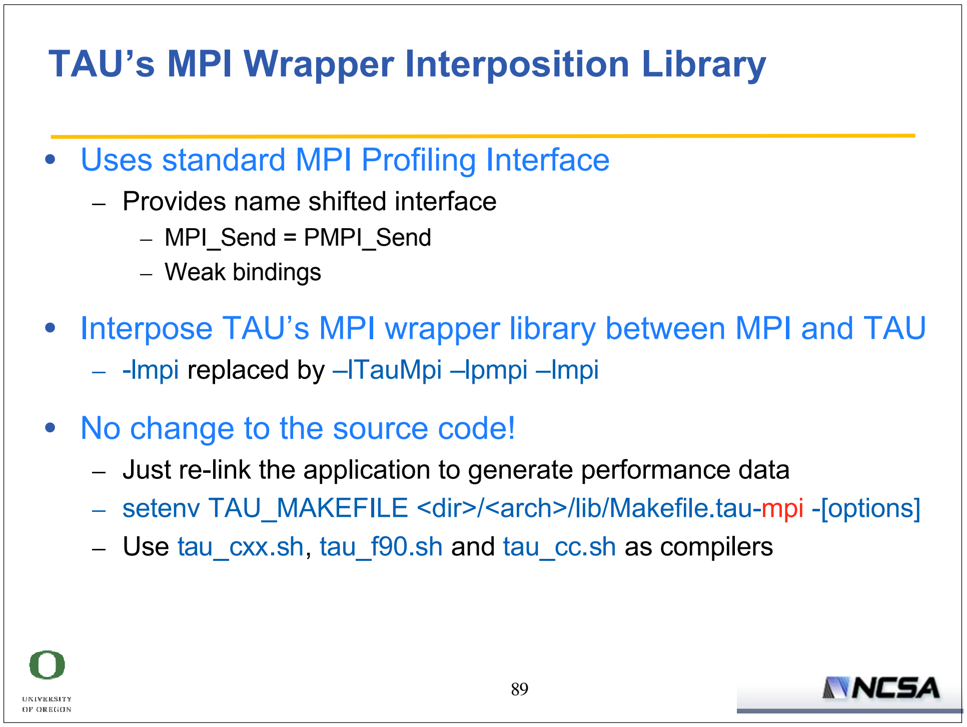

TAU: ParaProf Manager
File Options Help
Applications
Standard Applications
Default App
Default Exp
matmult/workshop/sameer/home/
Time
DB (jdbc:derby:/home/sameer/wyatt/tau2/i386_linux/lib/perfdmf)
TAU: ParaProf: /home/sameer/workshop/matmult
File Options Windows Help
Metric: Time
Value: Exclusive
std. dev.
mean
n,c,t 0,0,0
n,c,t 1,0,0
n,c,t 2,0,0
n,c,t 3,0,0
Show Thread Bar Chart
Show Thread Statistics Text Window
Show Thread Statistics Table
Show Thread Call Graph
Show Thread Call Path Relations
Show User Event Bar Chart
Show User Event Statistics Window
Add Thread to Comparison Window
TAU: ParaProf: n,c,t 0,0,0 – /home/sameer/workshop/matmult
File Options Windows Help

```
Metric: Time
Value: Exclusive
Units: microseconds

2096225  MPI_Send()
1339496  MPI_Init()
  76679  MPI_Bcast()
  74890  MAIN [{matmult.f90} {39,15}]
  60711  MPI_Recv()
  19583  INITIALIZE [{matmult.f90} {4,18}]
   6313  MPI_Finalize()
      4  MPI_Comm_rank()
      3  MPI_Comm_size()
```

click left mouse button

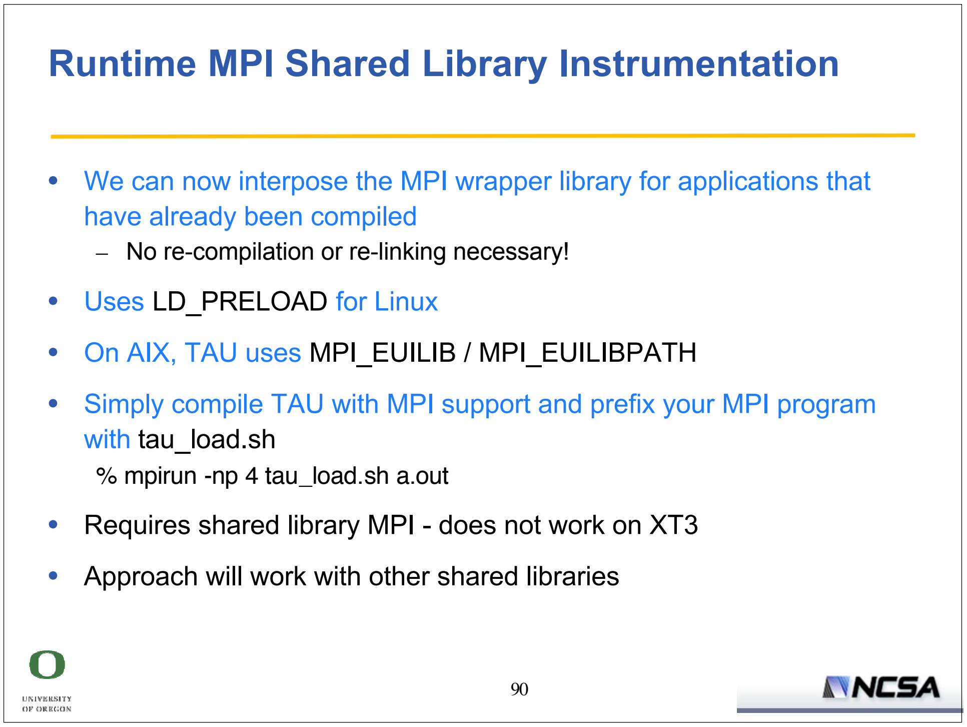
click right mouse button

```
% paraprof matmult.ppk
```



TAU’s MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface
    - MPI_Send = PMPI_Send
    - Weak bindings
- Interpose TAU’s MPI wrapper library between MPI and TAU
  - -lmpi replaced by –lTauMpi –lpmpi –lmpi
- No change to the source code!
  - Just re-link the application to generate performance data
  - setenv TAU_MAKEFILE <dir>/<arch>/lib/Makefile.tau-mpi -[options]
  - Use tau_cxx.sh, tau_f90.sh and tau_cc.sh as compilers



Runtime MPI Shared Library Instrumentation

- We can now interpose the MPI wrapper library for applications that have already been compiled
  - No re-compilation or re-linking necessary!
- Uses LD_PRELOAD for Linux
- On AIX, TAU uses MPI_EUILIB / MPI_EUILIBPATH
- Simply compile TAU with MPI support and prefix your MPI program with tau_load.sh

```
% mpirun -np 4 tau_load.sh a.out
```

- Requires shared library MPI - does not work on XT3
- Approach will work with other shared libraries

Instrumenting MPI Applications

- Under Linux you may use tau_load.sh to launch un-instrumented programs under TAU
  - Without TAU:

    % mpirun -np 4 ./a.out
  - With TAU:

    % ls /usr/pkgs/tau/x86_64/lib/libTAU*intel91*
    % mpirun -np 4 tau_load.sh ./a.out
    % mpirun -np 4 tau_load.sh -XrunTAUsh-mpi-pdt-trace.so a.out
    loads <taudir>/<arch>/lib/libTAUsh-mpi-pdt-trace.so shared object
- Under AIX, use tau_poe instead of poe
  - Without TAU:

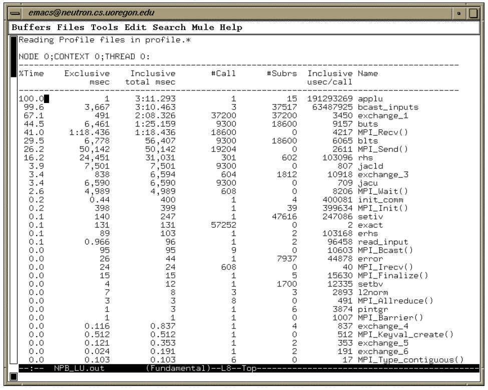
    % poe a.out -procs 8
  - With TAU:

    % tau_poe a.out -procs 8
    % tau_poe -XrunTAUsh-mpi-pdt-trace.so a.out -procs 8
    chooses <taudir>/<arch>/lib/libTAUsh-mpi-pdt-trace.so
- No change to source code or executables! No need to re-link!
- Only instruments MPI routines. To instrument user routines, you may need to parse the application source code!

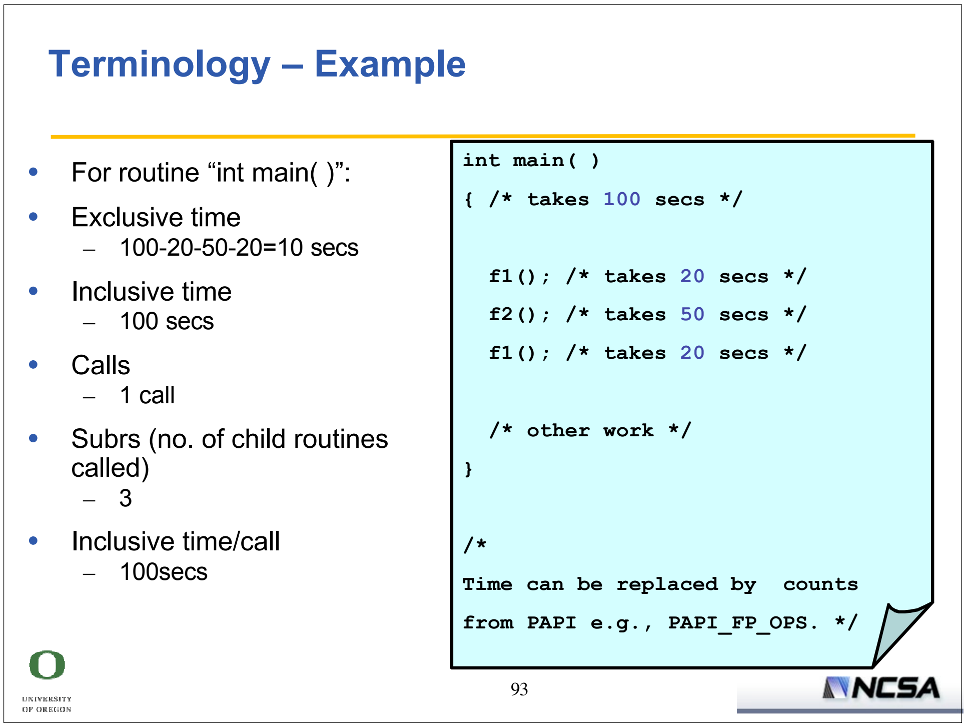
-PROFILE Configuration Option

- Generates flat profiles (one for each MPI process)
  - It is the default option.
- Uses wallclock time (gettimeofday() sys call)
- Calculates exclusive, inclusive time spent in each timer and number of calls

% pprof


emacs@neutron.cs.uoregon.edu
Buffers Files Tools Edit Search Mule Help

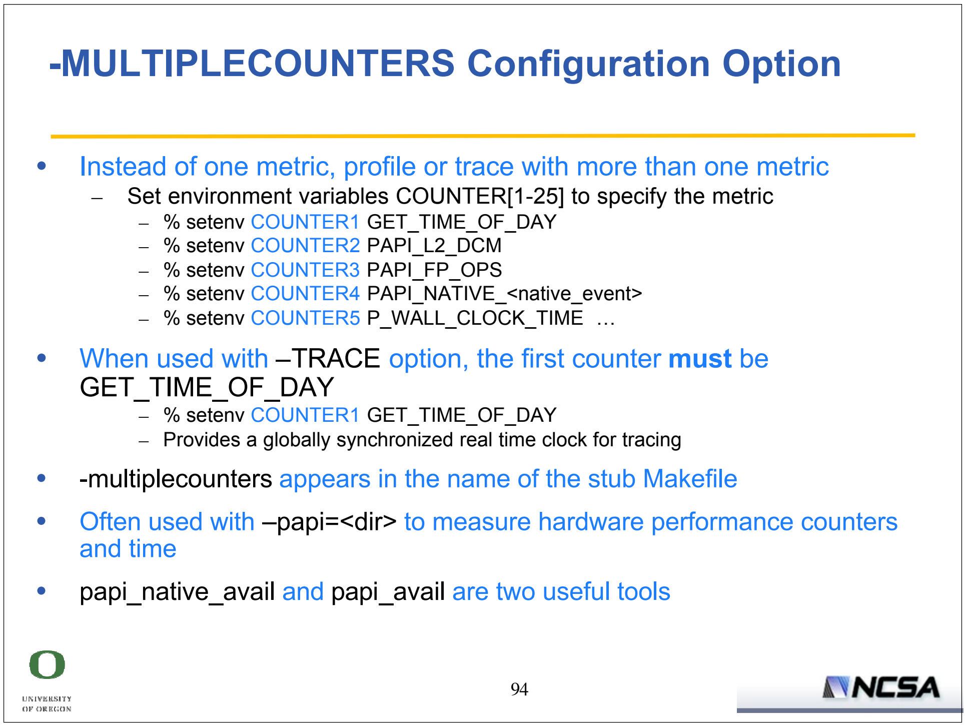
```
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0            1     3:11.293           1          15  191293269 applu
 99.6        3,667     3:10.463           3       37517   63487925 bcast_inputs
 67.1          491     2:08.326       37200       37200       3450 exchange_1
 44.5        6,461     1:25.159        9300       18600       9157 buts
 41.0     1:18.436     1:18.436       18600           0       4217 MPI_Recv()
 29.5        6,778       56,407        9300       18600       6065 blts
 26.2       50,142       50,142       19204           0       2611 MPI_Send()
 16.2       24,451       31,031         301         602     103096 rhs
  3.9        7,501        7,501        9300           0        807 jacld
  3.4          838        6,594         604        1812      10918 exchange_3
  3.4        6,590        6,590        9300           0        709 jacu
  2.6        4,989        4,989         608           0       8206 MPI_Wait()
  0.2         0.44          400           1           4     400081 init_comm
  0.2          398          399           1          39     399634 MPI_Init()
  0.1          140          247           1       47616     247086 setiv
  0.1          131          131       57252           0          2 exact
  0.1           89          103           1           2     103168 erhs
  0.1        0.966           96           1           2      96458 read_input
  0.0           95           95           9           0      10603 MPI_Bcast()
  0.0           26           44           1        7937      44878 error
  0.0           24           24         608           0         40 MPI_Irecv()
  0.0           15           15           1           5      15630 MPI_Finalize()
  0.0            4           12           1        1700      12335 setbv
  0.0            7            8           3           3       2893 l2norm
  0.0            3            3           8           0        491 MPI_Allreduce()
  0.0            1            3           1           6       3874 pintgr
  0.0            1            1           1           0       1007 MPI_Barrier()
  0.0        0.116        0.837           1           4        837 exchange_4
  0.0        0.512        0.512           1           0        512 MPI_Keyval_create()
  0.0        0.121        0.353           1           2        353 exchange_5
  0.0        0.024        0.191           1           2        191 exchange_6
  0.0        0.103        0.103           6           0         17 MPI_Type_contiguous()
--:--  NPB_LU.out       (Fundamental)--L8--Top------------------------------------------
```



Terminology – Example

- For routine “int main( )”:
- Exclusive time
  - 100-20-50-20=10 secs
- Inclusive time
  - 100 secs
- Calls
  - 1 call
- Subrs (no. of child routines called)
  - 3
- Inclusive time/call
  - 100secs

```
int main( )
{ /* takes 100 secs */

  f1(); /* takes 20 secs */
  f2(); /* takes 50 secs */
  f1(); /* takes 20 secs */

  /* other work */
}

/*
Time can be replaced by  counts
from PAPI e.g., PAPI_FP_OPS. */
```

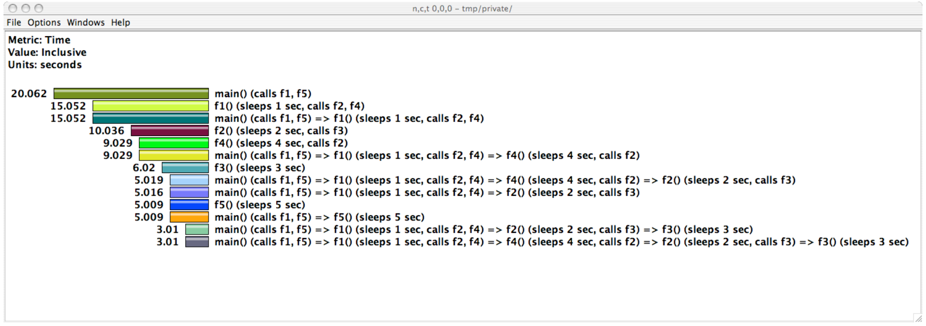


-MULTIPLECOUNTERS Configuration Option

- Instead of one metric, profile or trace with more than one metric
  - Set environment variables COUNTER[1-25] to specify the metric
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - % setenv COUNTER2 PAPI_L2_DCM
    - % setenv COUNTER3 PAPI_FP_OPS
    - % setenv COUNTER4 PAPI_NATIVE_<native_event>
    - % setenv COUNTER5 P_WALL_CLOCK_TIME …
- When used with –TRACE option, the first counter **must** be GET_TIME_OF_DAY
    - % setenv COUNTER1 GET_TIME_OF_DAY
    - Provides a globally synchronized real time clock for tracing
- -multiplecounters appears in the name of the stub Makefile
- Often used with –papi=<dir> to measure hardware performance counters and time
- papi_native_avail and papi_avail are two useful tools

-PROFILECALLPATH Configuration Option

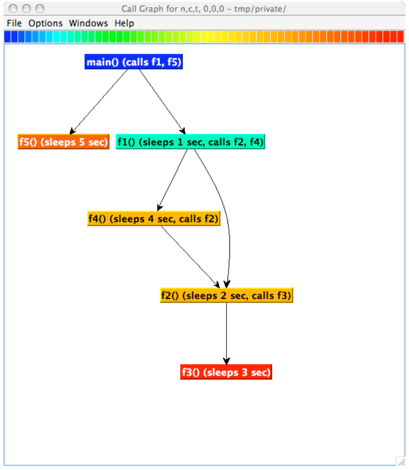
- Generates profiles that show the calling order (edges & nodes in callgraph)
  - A=>B=>C shows the time spent in C when it was called by B and B was called by A
  - Control the depth of callpath using TAU_CALLPATH_DEPTH env. Variable
  - -callpath in the name of the stub Makefile name

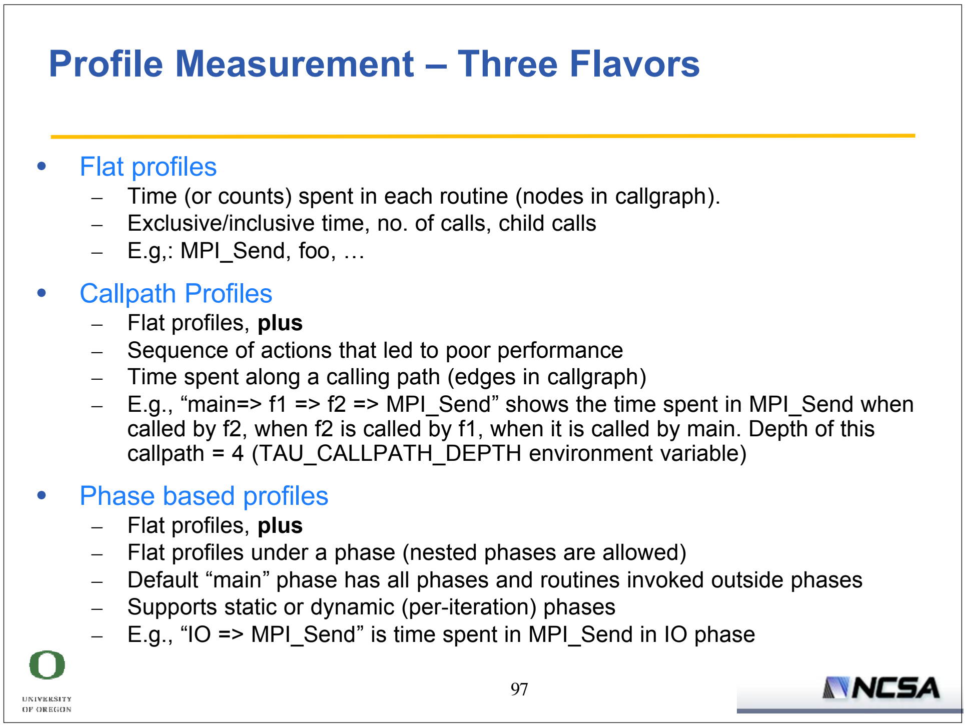


```
Metric: Time
Value: Inclusive
Units: seconds

20.062  main() (calls f1, f5)
15.052  f1() (sleeps 1 sec, calls f2, f4)
15.052  main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4)
10.036  f2() (sleeps 2 sec, calls f3)
 9.029  f4() (sleeps 4 sec, calls f2)
 9.029  main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f4() (sleeps 4 sec, calls f2)
  6.02  f3() (sleeps 3 sec)
 5.019  main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f4() (sleeps 4 sec, calls f2) => f2() (sleeps 2 sec, calls f3)
 5.016  main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f2() (sleeps 2 sec, calls f3)
 5.009  f5() (sleeps 5 sec)
 5.009  main() (calls f1, f5) => f5() (sleeps 5 sec)
  3.01  main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f2() (sleeps 2 sec, calls f3) => f3() (sleeps 3 sec)
  3.01  main() (calls f1, f5) => f1() (sleeps 1 sec, calls f2, f4) => f4() (sleeps 4 sec, calls f2) => f2() (sleeps 2 sec, calls f3) => f3() (sleeps 3 sec)
```

-PROFILECALLPATH Configuration Option

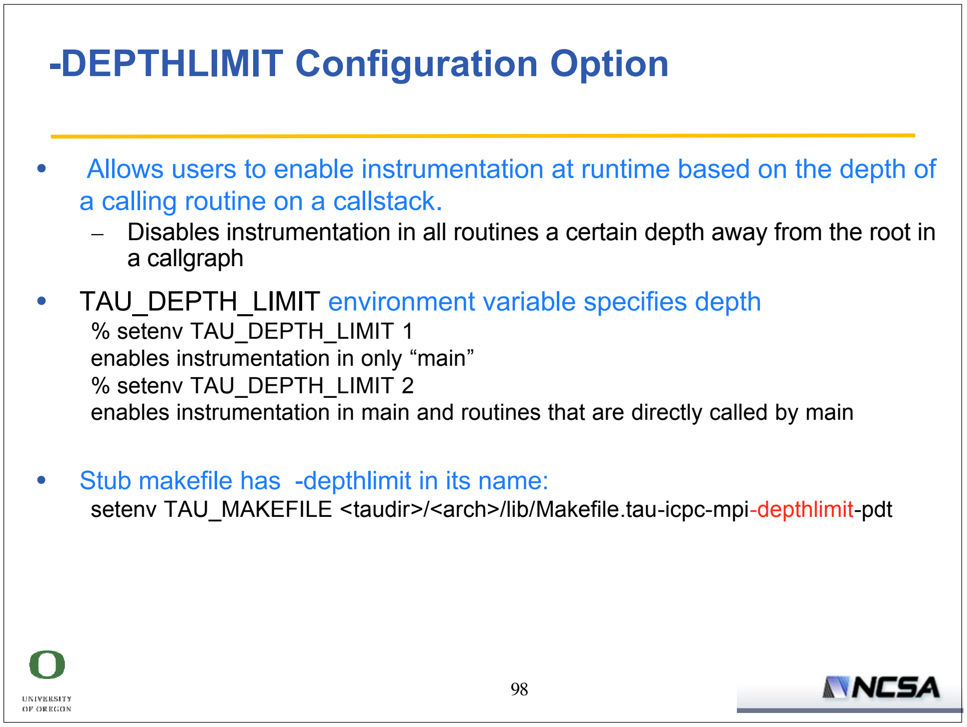
- Generates program callgraph


Call Graph for n,c,t, 0,0,0 – tmp/private/
File Options Windows Help
main() (calls f1, f5)
f5() (sleeps 5 sec)
f1() (sleeps 1 sec, calls f2, f4)
f4() (sleeps 4 sec, calls f2)
f2() (sleeps 2 sec, calls f3)
f3() (sleeps 3 sec)



Profile Measurement – Three Flavors

- Flat profiles
  - Time (or counts) spent in each routine (nodes in callgraph).
  - Exclusive/inclusive time, no. of calls, child calls
  - E.g,: MPI_Send, foo, …
- Callpath Profiles
  - Flat profiles, **plus**
  - Sequence of actions that led to poor performance
  - Time spent along a calling path (edges in callgraph)
  - E.g., “main=> f1 => f2 => MPI_Send” shows the time spent in MPI_Send when called by f2, when f2 is called by f1, when it is called by main. Depth of this callpath = 4 (TAU_CALLPATH_DEPTH environment variable)
- Phase based profiles
  - Flat profiles, **plus**
  - Flat profiles under a phase (nested phases are allowed)
  - Default “main” phase has all phases and routines invoked outside phases
  - Supports static or dynamic (per-iteration) phases
  - E.g., “IO => MPI_Send” is time spent in MPI_Send in IO phase



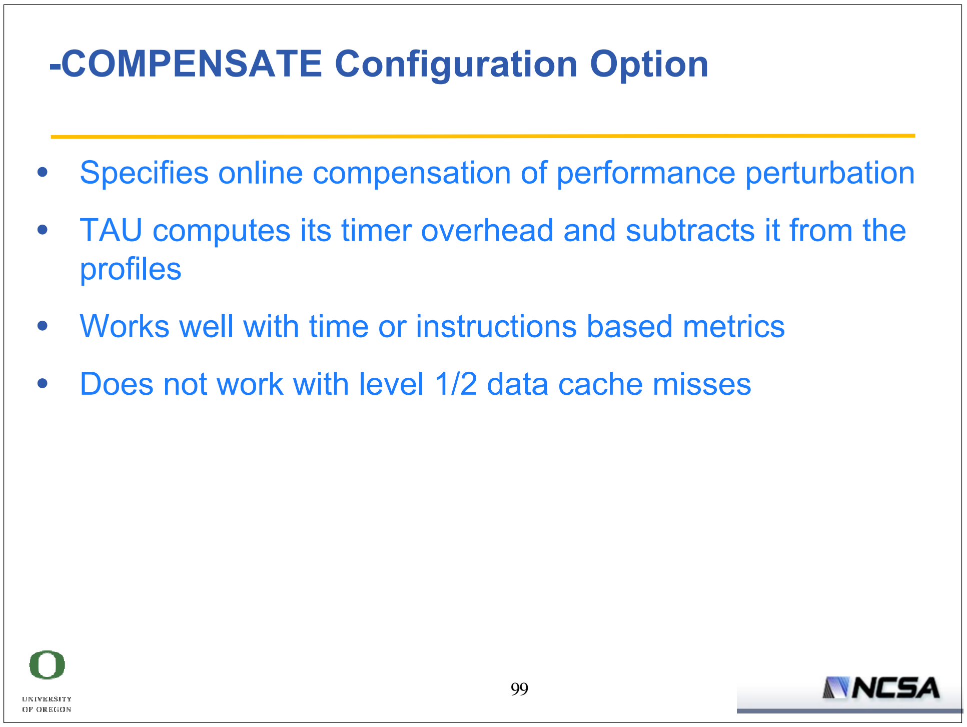
-DEPTHLIMIT Configuration Option

- Allows users to enable instrumentation at runtime based on the depth of a calling routine on a callstack.
  - Disables instrumentation in all routines a certain depth away from the root in a callgraph
- TAU_DEPTH_LIMIT environment variable specifies depth

```
% setenv TAU_DEPTH_LIMIT 1
enables instrumentation in only “main”
% setenv TAU_DEPTH_LIMIT 2
enables instrumentation in main and routines that are directly called by main
```

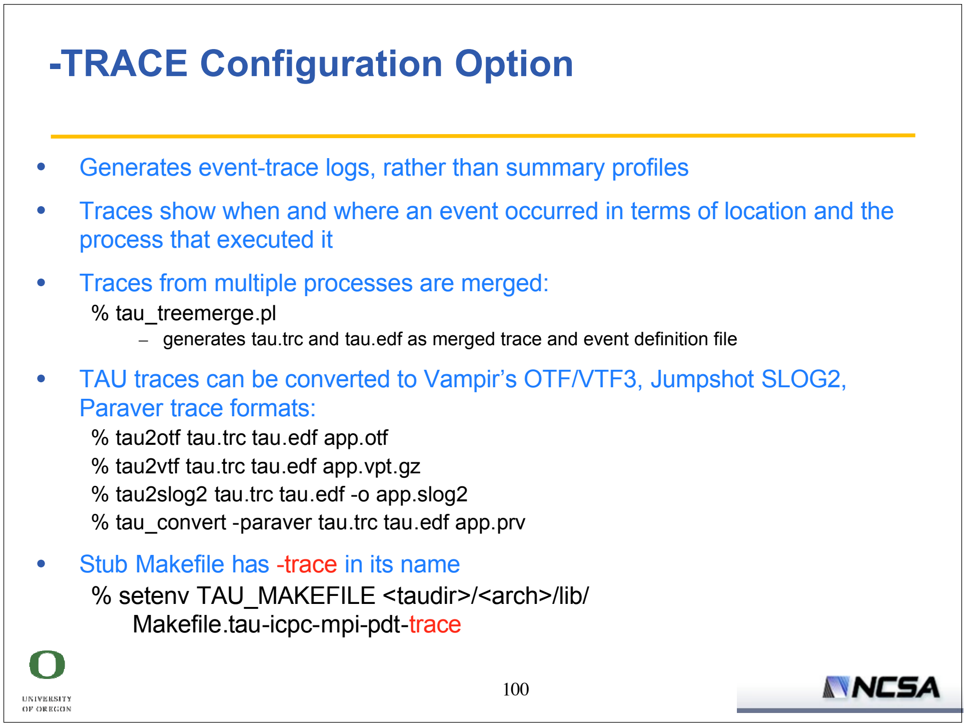
- Stub makefile has -depthlimit in its name:

```
setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-mpi-depthlimit-pdt
```



-COMPENSATE Configuration Option

- Specifies online compensation of performance perturbation
- TAU computes its timer overhead and subtracts it from the profiles
- Works well with time or instructions based metrics
- Does not work with level 1/2 data cache misses



-TRACE Configuration Option

- Generates event-trace logs, rather than summary profiles
- Traces show when and where an event occurred in terms of location and the process that executed it
- Traces from multiple processes are merged:

```
% tau_treemerge.pl
```

  – generates tau.trc and tau.edf as merged trace and event definition file

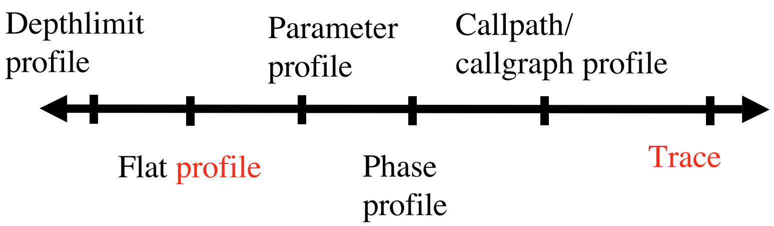
- TAU traces can be converted to Vampir's OTF/VTF3, Jumpshot SLOG2, Paraver trace formats:

```
% tau2otf tau.trc tau.edf app.otf
% tau2vtf tau.trc tau.edf app.vpt.gz
% tau2slog2 tau.trc tau.edf -o app.slog2
% tau_convert -paraver tau.trc tau.edf app.prv
```

- Stub Makefile has -trace in its name

```
% setenv TAU_MAKEFILE <taudir>/<arch>/lib/
      Makefile.tau-icpc-mpi-pdt-trace
```

Performance Evaluation Alternatives



Each alternative has:
- one metric/counter
- multiple counters

Volume of performance data

-PROFILEPARAM Configuration Option

- Idea: partition performance data for individual functions based on runtime parameters
- Enable by configuring with –PROFILEPARAM
- TAU call: TAU_PROFILE_PARAM1L (value, “name”)
- Simple example:

```
void foo(long input) {
   TAU_PROFILE("foo", "", TAU_DEFAULT);
   TAU_PROFILE_PARAM1L(input, "input");
   ... }
```

Workload Characterization

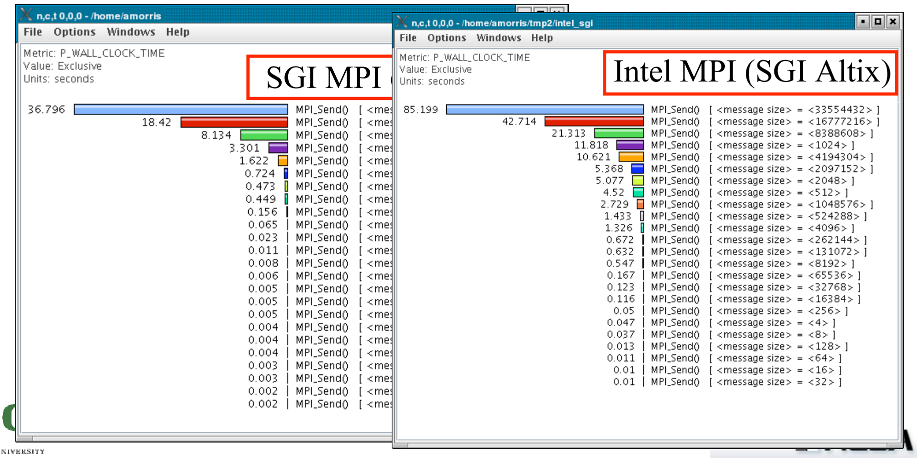
- 5 seconds spent in function “`foo`” becomes
  - 2 seconds for “`foo [ <input> = <25> ]`”
  - 1 seconds for “`foo [ <input> = <5> ]`”
  - …
- Currently used in MPI wrapper library
  - Allows for partitioning of time spent in MPI routines based on parameters (message size, message tag, destination node)
  - Can be extrapolated to infer specifics about the MPI subsystem and system as a whole

Workload Characterization

```
#include <stdio.h>
#include <mpi.h>
int buffer[8*1024*1024];

int main(int argc, char **argv) {
  int rank, size, i, j;
  MPI_Init(&argc, &argv);
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  for (i=0;i<1000;i++)
    for (j=1;j<=8*1024*1024;j*=2) {
      if (rank == 0) {
        MPI_Send(buffer,j,MPI_INT,1,42,MPI_COMM_WORLD);
      } else {
        MPI_Status status;
        MPI_Recv(buffer,j,MPI_INT,0,42,MPI_COMM_WORLD,&status);
      }
    }
  MPI_Finalize();
}
```

Workload Characterization

```
% icc mpi.c -lmpi
% mpirun -np 2 tau_load.sh -XrunTAU-icpc-mpi-pdt.so a.out
```



SGI MPI

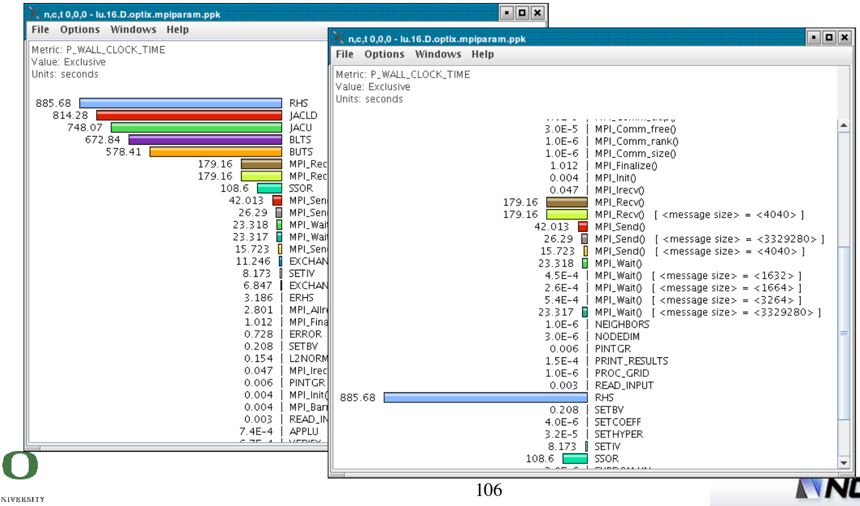
Intel MPI (SGI Altix)

```
n,c,t 0,0,0 - /home/amorris
File  Options  Windows  Help
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

36.796  MPI_Send()  [ <mes
18.42   MPI_Send()  [ <mes
8.134   MPI_Send()  [ <mes
3.301   MPI_Send()  [ <mes
1.622   MPI_Send()  [ <mes
0.724   MPI_Send()  [ <mes
0.473   MPI_Send()  [ <mes
0.449   MPI_Send()  [ <mes
0.156   MPI_Send()  [ <mes
0.065   MPI_Send()  [ <mes
0.023   MPI_Send()  [ <mes
0.011   MPI_Send()  [ <mes
0.008   MPI_Send()  [ <mes
0.006   MPI_Send()  [ <mes
0.005   MPI_Send()  [ <mes
0.005   MPI_Send()  [ <mes
0.005   MPI_Send()  [ <mes
0.004   MPI_Send()  [ <mes
0.004   MPI_Send()  [ <mes
0.004   MPI_Send()  [ <mes
0.003   MPI_Send()  [ <mes
0.003   MPI_Send()  [ <mes
0.002   MPI_Send()  [ <mes
0.002   MPI_Send()  [ <mes
```

```
n,c,t 0,0,0 - /home/amorris/tmp2/intel_sgi
File  Options  Windows  Help
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

85.199  MPI_Send()  [ <message size> = <33554432> ]
42.714  MPI_Send()  [ <message size> = <16777216> ]
21.313  MPI_Send()  [ <message size> = <8388608> ]
11.818  MPI_Send()  [ <message size> = <1024> ]
10.621  MPI_Send()  [ <message size> = <4194304> ]
5.368   MPI_Send()  [ <message size> = <2097152> ]
5.077   MPI_Send()  [ <message size> = <2048> ]
4.52    MPI_Send()  [ <message size> = <512> ]
2.729   MPI_Send()  [ <message size> = <1048576> ]
1.433   MPI_Send()  [ <message size> = <524288> ]
1.326   MPI_Send()  [ <message size> = <4096> ]
0.672   MPI_Send()  [ <message size> = <262144> ]
0.632   MPI_Send()  [ <message size> = <131072> ]
0.547   MPI_Send()  [ <message size> = <8192> ]
0.167   MPI_Send()  [ <message size> = <65536> ]
0.123   MPI_Send()  [ <message size> = <32768> ]
0.116   MPI_Send()  [ <message size> = <16384> ]
0.05    MPI_Send()  [ <message size> = <256> ]
0.047   MPI_Send()  [ <message size> = <4> ]
0.037   MPI_Send()  [ <message size> = <8> ]
0.013   MPI_Send()  [ <message size> = <128> ]
0.011   MPI_Send()  [ <message size> = <64> ]
0.01    MPI_Send()  [ <message size> = <16> ]
0.01    MPI_Send()  [ <message size> = <32> ]
```

Workload Characterization

- MPI Results (NAS Parallel Benchmark 3.1, LU class D on



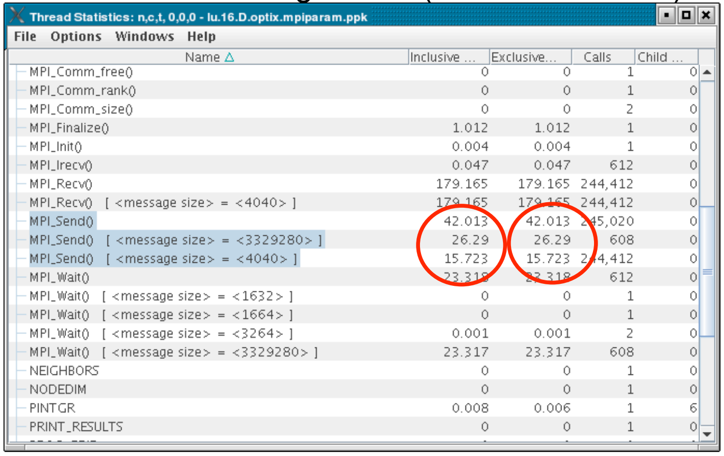
```
n,c,t 0,0,0 - lu.16.D.optix.mpiparam.ppk
File  Options  Windows  Help
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

885.68  RHS
814.28  JACLD
748.07  JACU
672.84  BLTS
578.41  BUTS
179.16  MPI_Rec
179.16  MPI_Rec
108.6   SSOR
42.013  MPI_Sen
26.29   MPI_Sen
23.318  MPI_Wai
23.317  MPI_Wai
15.723  MPI_Sen
11.246  EXCHAN
8.173   SETIV
6.847   EXCHAN
3.186   ERHS
2.801   MPI_Allr
1.012   MPI_Fina
0.728   ERROR
0.208   SETBV
0.154   L2NORM
0.047   MPI_Irec
0.006   PINTGR
0.004   MPI_Init(
0.004   MPI_Barr
0.003   READ_IN
7.4E-4  APPLU
```

```
n,c,t 0,0,0 - lu.16.D.optix.mpiparam.ppk
File  Options  Windows  Help
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

3.0E-5  MPI_Comm_free()
1.0E-6  MPI_Comm_rank()
1.0E-6  MPI_Comm_size()
1.012   MPI_Finalize()
0.004   MPI_Init()
0.047   MPI_Irecv()
179.16  MPI_Recv()
179.16  MPI_Recv()  [ <message size> = <4040> ]
42.013  MPI_Send()
26.29   MPI_Send()  [ <message size> = <3329280> ]
15.723  MPI_Send()  [ <message size> = <4040> ]
23.318  MPI_Wait()
4.5E-4  MPI_Wait()  [ <message size> = <1632> ]
2.6E-4  MPI_Wait()  [ <message size> = <1664> ]
5.4E-4  MPI_Wait()  [ <message size> = <3264> ]
23.317  MPI_Wait()  [ <message size> = <3329280> ]
1.0E-6  NEIGHBORS
3.0E-6  NODEDIM
0.006   PINTGR
1.5E-4  PRINT_RESULTS
1.0E-6  PROC_GRID
0.003   READ_INPUT
885.68  RHS
0.208   SETBV
4.0E-6  SETCOEFF
3.2E-5  SETHYPER
8.173   SETIV
108.6   SSOR
```

106

Workload Characterization

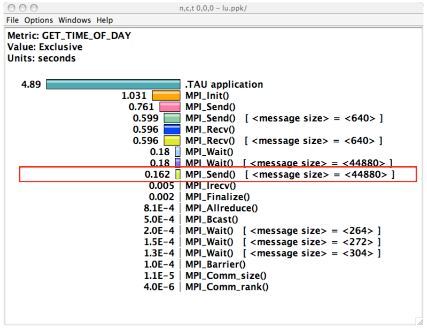
- Two different message sizes (~3.3MB and ~4K)



Thread Statistics: n,c,t, 0,0,0 - lu.16.D.optix.mpiparam.ppk
File Options Windows Help

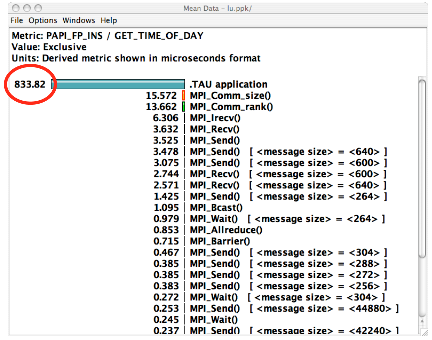
Name △	Inclusive ...	Exclusive...	Calls	Child ...
MPI_Comm_free()	0	0	1	0
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	2	0
MPI_Finalize()	1.012	1.012	1	0
MPI_Init()	0.004	0.004	1	0
MPI_Irecv()	0.047	0.047	612	0
MPI_Recv()	179.165	179.165	244,412	0
MPI_Recv() [<message size> = <4040>]	179.165	179.165	244,412	0
MPI_Send()	42.013	42.013	245,020	0
MPI_Send() [<message size> = <3329280>]	26.29	26.29	608	0
MPI_Send() [<message size> = <4040>]	15.723	15.723	244,412	0
MPI_Wait()	23.318	23.318	612	0
MPI_Wait() [<message size> = <1632>]	0	0	1	0
MPI_Wait() [<message size> = <1664>]	0	0	1	0
MPI_Wait() [<message size> = <3264>]	0.001	0.001	2	0
MPI_Wait() [<message size> = <3329280>]	23.317	23.317	608	0
NEIGHBORS	0	0	1	0
NODEDIM	0	0	1	0
PINTGR	0.008	0.006	1	6
PRINT_RESULTS	0	0	1	0

Job Tracking: ParaProf profile browser



Metric: GET_TIME_OF_DAY
Value: Exclusive
Units: seconds

```
4.89     .TAU application
1.031    MPI_Init()
0.761    MPI_Send()
0.599    MPI_Send()   [ <message size> = <640> ]
0.596    MPI_Recv()
0.596    MPI_Recv()   [ <message size> = <640> ]
0.18     MPI_Wait()
0.18     MPI_Wait()   [ <message size> = <44880> ]
0.162    MPI_Send()   [ <message size> = <44880> ]
0.005    MPI_Irecv()
0.002    MPI_Finalize()
8.1E-4   MPI_Allreduce()
5.0E-4   MPI_Bcast()
2.0E-4   MPI_Wait()   [ <message size> = <264> ]
1.5E-4   MPI_Wait()   [ <message size> = <272> ]
1.3E-4   MPI_Wait()   [ <message size> = <304> ]
1.0E-4   MPI_Barrier()
1.1E-5   MPI_Comm_size()
4.0E-6   MPI_Comm_rank()
```



Metric: PAPI_FP_INS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format

```
833.82   .TAU application
15.572   MPI_Comm_size()
13.662   MPI_Comm_rank()
6.306    MPI_Irecv()
3.632    MPI_Recv()
3.525    MPI_Send()
3.478    MPI_Send()   [ <message size> = <640> ]
3.075    MPI_Send()   [ <message size> = <600> ]
2.744    MPI_Recv()   [ <message size> = <600> ]
2.571    MPI_Recv()   [ <message size> = <640> ]
1.425    MPI_Send()   [ <message size> = <264> ]
1.095    MPI_Bcast()
0.979    MPI_Wait()   [ <message size> = <264> ]
0.853    MPI_Allreduce()
0.715    MPI_Barrier()
0.467    MPI_Send()   [ <message size> = <304> ]
0.385    MPI_Send()   [ <message size> = <288> ]
0.385    MPI_Send()   [ <message size> = <272> ]
0.383    MPI_Send()   [ <message size> = <256> ]
0.272    MPI_Wait()   [ <message size> = <304> ]
0.253    MPI_Send()   [ <message size> = <44880> ]
0.245    MPI_Wait()
0.237    MPI_Send()   [ <message size> = <42240> ]
```

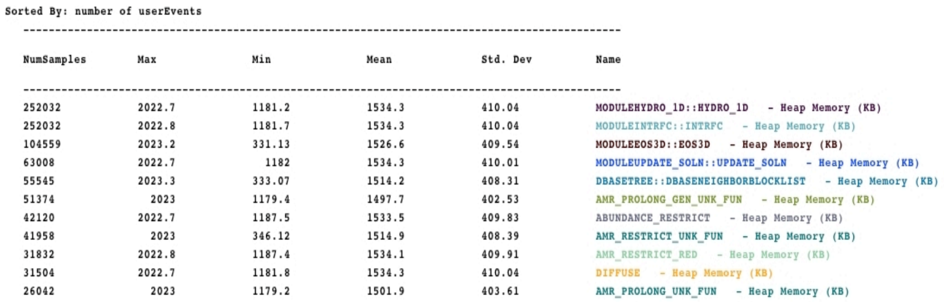
LU spent 0.162 seconds sending messages of size 44880

It got 833.82 Mflops!

Memory Profiling in TAU

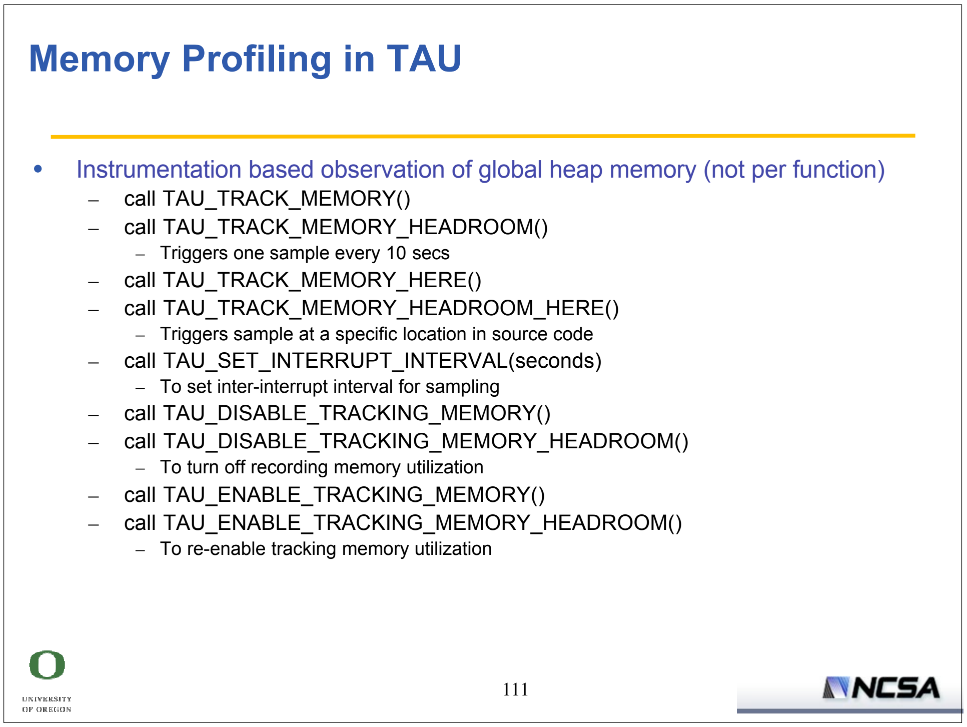
- Configuration option –PROFILEMEMORY
  - Records global heap memory utilization for each function
  - Takes one sample at beginning of each function and associates the sample with function name
- Configuration option -PROFILEHEADROOM
  - Records headroom (amount of free memory to grow) for each function
  - Takes one sample at beginning of each function and associates it with the callstack [TAU_CALLPATH_DEPTH env variable]
  - Useful for debugging memory usage on IBM BG/L.
- Independent of instrumentation/measurement options selected
- No need to insert macros/calls in the source code
- User defined atomic events appear in profiles/traces

Memory Profiling in TAU (Atomic events)



```
Sorted By: number of userEvents
   ---------------------------------------------------------------------------------------------

   NumSamples        Max              Min              Mean             Std. Dev         Name

   ---------------------------------------------------------------------------------------------
   252032            2022.7           1181.2           1534.3           410.04           MODULEHYDRO_1D::HYDRO_1D   - Heap Memory (KB)
   252032            2022.8           1181.7           1534.3           410.04           MODULEINTRFC::INTRFC   - Heap Memory (KB)
   104559            2023.2           331.13           1526.6           409.54           MODULEEOS3D::EOS3D   - Heap Memory (KB)
   63008             2022.7             1182           1534.3           410.01           MODULEUPDATE_SOLN::UPDATE_SOLN   - Heap Memory (KB)
   55545             2023.3           333.07           1514.2           408.31           DBASETREE::DBASENEIGHBORBLOCKLIST   - Heap Memory (KB)
   51374               2023           1179.4           1497.7           402.53           AMR_PROLONG_GEN_UNK_FUN   - Heap Memory (KB)
   42120             2022.7           1187.5           1533.5           409.83           ABUNDANCE_RESTRICT   - Heap Memory (KB)
   41958               2023           346.12           1514.9           408.39           AMR_RESTRICT_UNK_FUN   - Heap Memory (KB)
   31832             2022.8           1187.4           1534.1           409.91           AMR_RESTRICT_RED   - Heap Memory (KB)
   31504             2022.7           1181.8           1534.3           410.04           DIFFUSE   - Heap Memory (KB)
   26042               2023           1179.2           1501.9           403.61           AMR_PROLONG_UNK_FUN   - Heap Memory (KB)
```

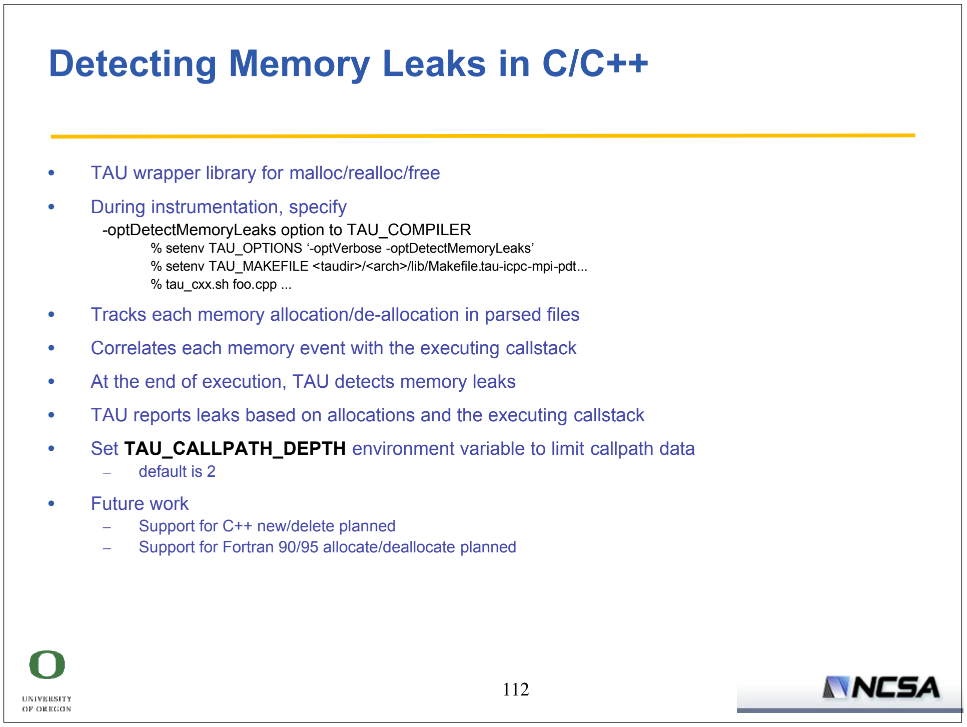
Flash2 code profile (-PROFILEMEMORY) on IBM BlueGene/L [MPI rank 0]



Memory Profiling in TAU

- Instrumentation based observation of global heap memory (not per function)
  - call TAU_TRACK_MEMORY()
  - call TAU_TRACK_MEMORY_HEADROOM()
    - Triggers one sample every 10 secs
  - call TAU_TRACK_MEMORY_HERE()
  - call TAU_TRACK_MEMORY_HEADROOM_HERE()
    - Triggers sample at a specific location in source code
  - call TAU_SET_INTERRUPT_INTERVAL(seconds)
    - To set inter-interrupt interval for sampling
  - call TAU_DISABLE_TRACKING_MEMORY()
  - call TAU_DISABLE_TRACKING_MEMORY_HEADROOM()
    - To turn off recording memory utilization
  - call TAU_ENABLE_TRACKING_MEMORY()
  - call TAU_ENABLE_TRACKING_MEMORY_HEADROOM()
    - To re-enable tracking memory utilization

111



Detecting Memory Leaks in C/C++

- TAU wrapper library for malloc/realloc/free
- During instrumentation, specify

  -optDetectMemoryLeaks option to TAU_COMPILER

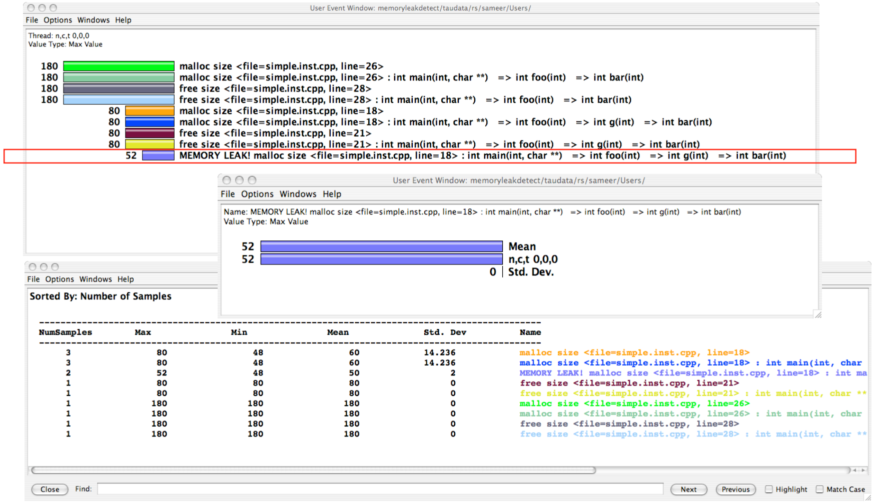
  ```
  % setenv TAU_OPTIONS '-optVerbose -optDetectMemoryLeaks'
  % setenv TAU_MAKEFILE <taudir>/<arch>/lib/Makefile.tau-icpc-mpi-pdt...
  % tau_cxx.sh foo.cpp ...
  ```
- Tracks each memory allocation/de-allocation in parsed files
- Correlates each memory event with the executing callstack
- At the end of execution, TAU detects memory leaks
- TAU reports leaks based on allocations and the executing callstack
- Set **TAU_CALLPATH_DEPTH** environment variable to limit callpath data
  - default is 2
- Future work
  - Support for C++ new/delete planned
  - Support for Fortran 90/95 allocate/deallocate planned

112

Detecting Memory Leaks in C/C++

```
include /opt/tau/x86_64/lib/Makefile.tau-icpc-mpi-pdt
MYOPTS = -optVerbose -optDetectMemoryLeaks
CC= $(TAU_COMPILER) $(MYOPTS) $(TAU_CXX)
LIBS = -lm
OBJS = f1.o f2.o ...
TARGET= a.out
TARGET: $(OBJS)
	$(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.c.o:
	$(CC) $(CFLAGS) -c $< -o $@
```

Memory Leak Detection



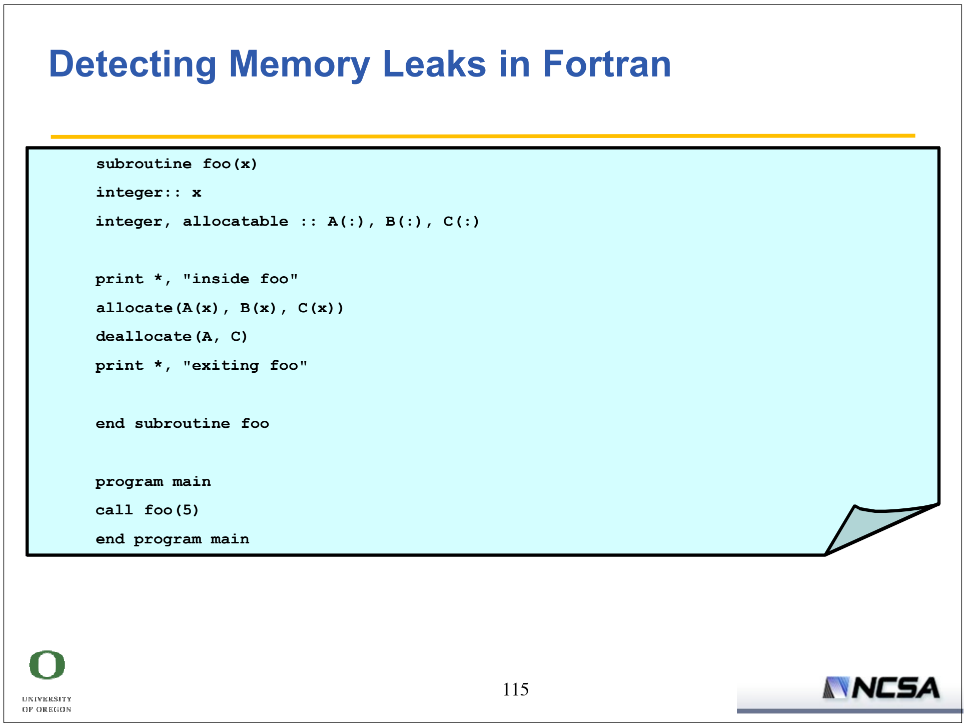
User Event Window: memoryleakdetect/taudata/rs/sameer/Users/

```
Thread: n,c,t 0,0,0
Value Type: Max Value

180  malloc size <file=simple.inst.cpp, line=26>
180  malloc size <file=simple.inst.cpp, line=26> : int main(int, char **)   => int foo(int)   => int bar(int)
180  free size <file=simple.inst.cpp, line=28>
180  free size <file=simple.inst.cpp, line=28> : int main(int, char **)   => int foo(int)   => int bar(int)
 80  malloc size <file=simple.inst.cpp, line=18>
 80  malloc size <file=simple.inst.cpp, line=18> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
 80  free size <file=simple.inst.cpp, line=21>
 80  free size <file=simple.inst.cpp, line=21> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
 52  MEMORY LEAK! malloc size <file=simple.inst.cpp, line=18> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
```

User Event Window: memoryleakdetect/taudata/rs/sameer/Users/

```
Name: MEMORY LEAK! malloc size <file=simple.inst.cpp, line=18> : int main(int, char **)   => int foo(int)   => int g(int)   => int bar(int)
Value Type: Max Value

52  Mean
52  n,c,t 0,0,0
 0  Std. Dev.
```

```
Sorted By: Number of Samples

---------------------------------------------------------------------------------------
NumSamples     Max         Min         Mean        Std. Dev     Name
---------------------------------------------------------------------------------------
     3          80          48          60          14.236      malloc size <file=simple.inst.cpp, line=18>
     3          80          48          60          14.236      malloc size <file=simple.inst.cpp, line=18> : int main(int, char
     2          52          48          50               2      MEMORY LEAK! malloc size <file=simple.inst.cpp, line=18> : int ma
     1          80          80          80               0      free size <file=simple.inst.cpp, line=21>
     1          80          80          80               0      free size <file=simple.inst.cpp, line=21> : int main(int, char **
     1         180         180         180               0      malloc size <file=simple.inst.cpp, line=26>
     1         180         180         180               0      malloc size <file=simple.inst.cpp, line=26> : int main(int, char
     1         180         180         180               0      free size <file=simple.inst.cpp, line=28>
     1         180         180         180               0      free size <file=simple.inst.cpp, line=28> : int main(int, char **
```

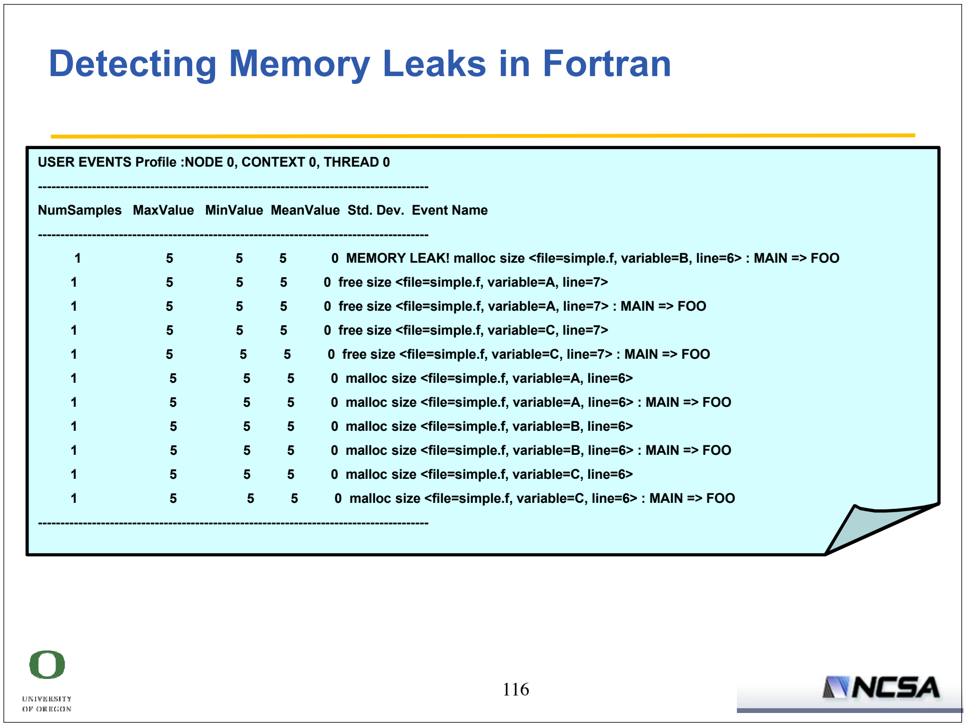


Detecting Memory Leaks in Fortran

```
subroutine foo(x)
integer:: x
integer, allocatable :: A(:), B(:), C(:)

print *, "inside foo"
allocate(A(x), B(x), C(x))
deallocate(A, C)
print *, "exiting foo"

end subroutine foo

program main
call foo(5)
end program main
```

UNIVERSITY OF OREGON

NCSA



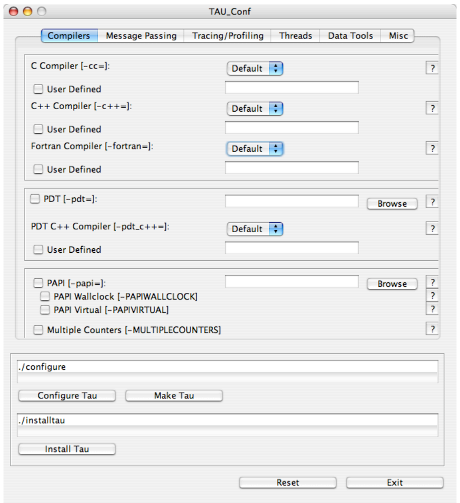
Detecting Memory Leaks in Fortran

```
USER EVENTS Profile :NODE 0, CONTEXT 0, THREAD 0
---------------------------------------------------------------------------------------
NumSamples   MaxValue   MinValue   MeanValue   Std. Dev.   Event Name
---------------------------------------------------------------------------------------
      1          5          5          5          0  MEMORY LEAK! malloc size <file=simple.f, variable=B, line=6> : MAIN => FOO
      1          5          5          5          0  free size <file=simple.f, variable=A, line=7>
      1          5          5          5          0  free size <file=simple.f, variable=A, line=7> : MAIN => FOO
      1          5          5          5          0  free size <file=simple.f, variable=C, line=7>
      1          5          5          5          0  free size <file=simple.f, variable=C, line=7> : MAIN => FOO
      1          5          5          5          0  malloc size <file=simple.f, variable=A, line=6>
      1          5          5          5          0  malloc size <file=simple.f, variable=A, line=6> : MAIN => FOO
      1          5          5          5          0  malloc size <file=simple.f, variable=B, line=6>
      1          5          5          5          0  malloc size <file=simple.f, variable=B, line=6> : MAIN => FOO
      1          5          5          5          0  malloc size <file=simple.f, variable=C, line=6>
      1          5          5          5          0  malloc size <file=simple.f, variable=C, line=6> : MAIN => FOO
---------------------------------------------------------------------------------------
```

UNIVERSITY OF OREGON

NCSA

TAU_SETUP: A GUI for Installing TAU



C Compiler [-cc=]: Default ?

□ User Defined

C++ Compiler [-c++=]: Default ?

□ User Defined

Fortran Compiler [-fortran=]: Default ?

□ User Defined

□ PDT [-pdt=]: Browse ?

PDT C++ Compiler [-pdt_c++=]: Default ?

□ User Defined

□ PAPI [-papi=]: Browse ?

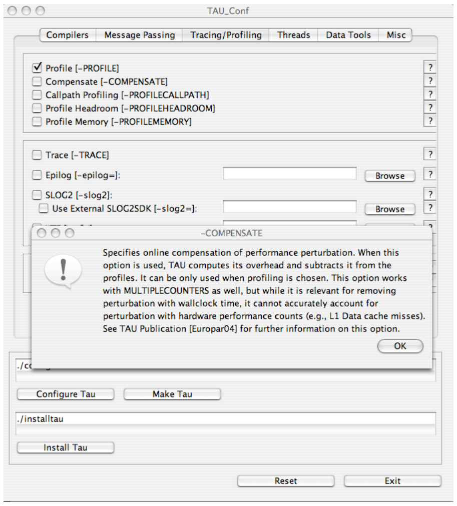
□ PAPI Wallclock [-PAPIWALLCLOCK] ?

□ PAPI Virtual [-PAPIVIRTUAL] ?

□ Multiple Counters [-MULTIPLECOUNTERS] ?

```
./configure
```

```
./installtau
```



☑ Profile [-PROFILE] ?

□ Compensate [-COMPENSATE] ?

□ Callpath Profiling [-PROFILECALLPATH] ?

□ Profile Headroom [-PROFILEHEADROOM] ?

□ Profile Memory [-PROFILEMEMORY] ?

□ Trace [-TRACE] ?

□ Epilog [-epilog=]: Browse ?

□ SLOG2 [-slog2]: ?

□ Use External SLOG2SDK [-slog2=]: Browse ?

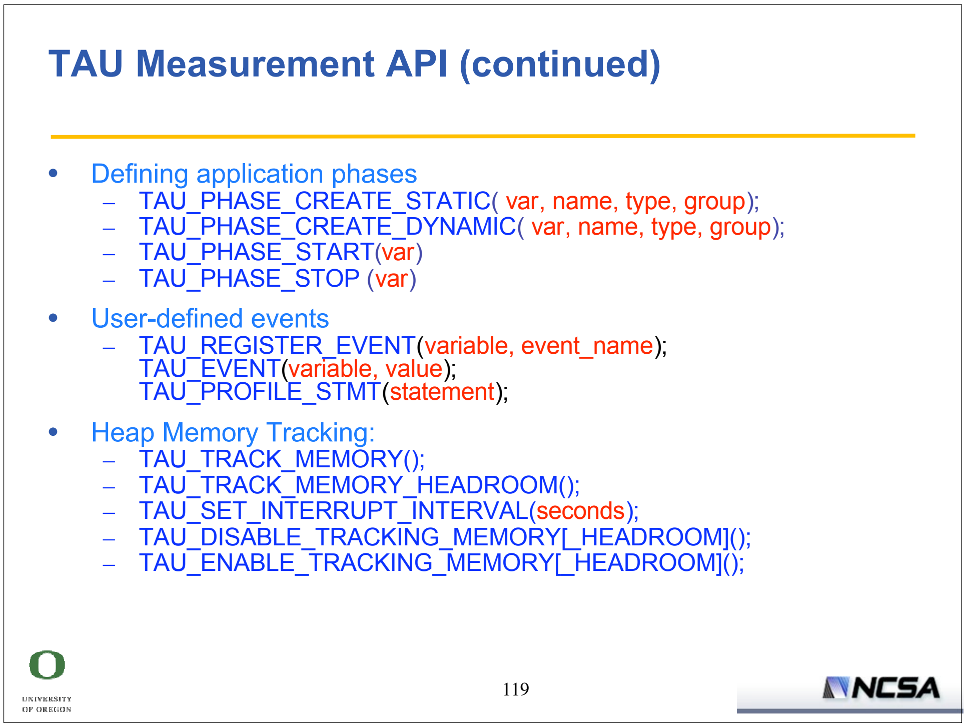
-COMPENSATE

Specifies online compensation of performance perturbation. When this option is used, TAU computes its overhead and subtracts it from the profiles. It can be only used when profiling is chosen. This option works with MULTIPLECOUNTERS as well, but while it is relevant for removing perturbation with wallclock time, it cannot accurately account for perturbation with hardware performance counts (e.g., L1 Data cache misses). See TAU Publication [Europar04] for further information on this option.

```
./installtau
```

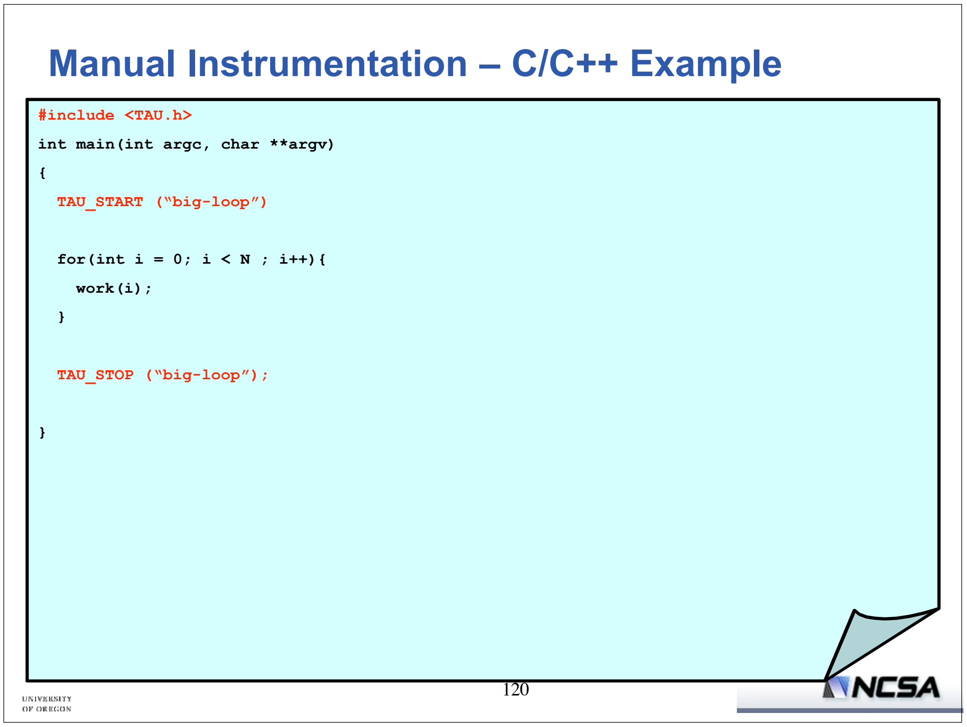
TAU Manual Instrumentation API for C/C++

- Initialization and runtime configuration
  - TAU_PROFILE_INIT(argc, argv);
    TAU_PROFILE_SET_NODE(myNode);
    TAU_PROFILE_SET_CONTEXT(myContext);
    TAU_PROFILE_EXIT(message);
    TAU_REGISTER_THREAD();
- Function and class methods for C++ only:
  - TAU_PROFILE(name, type, group);
  - TAU_PROFILE (name, type, group);
- Name-based API
  - TAU_START("timer_name");
    TAU_STOP("timer_name");
- User-defined timing
  - TAU_PROFILE_TIMER(timer, name, type, group);
    TAU_PROFILE_START(timer);
    TAU_PROFILE_STOP(timer);



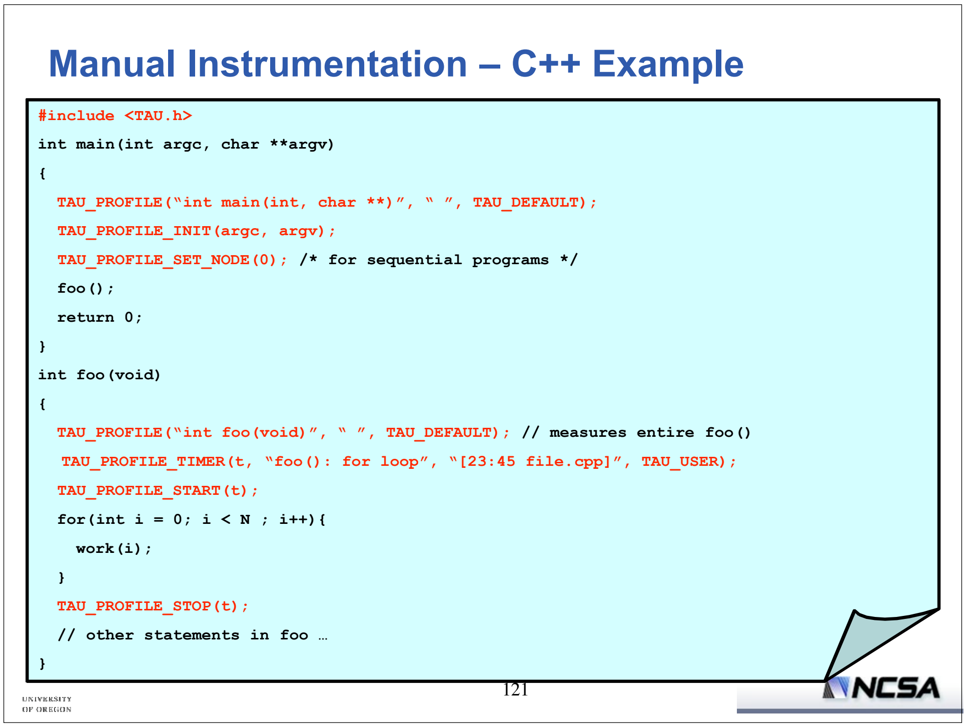
TAU Measurement API (continued)

- Defining application phases
  - TAU_PHASE_CREATE_STATIC(var, name, type, group);
  - TAU_PHASE_CREATE_DYNAMIC(var, name, type, group);
  - TAU_PHASE_START(var)
  - TAU_PHASE_STOP (var)
- User-defined events
  - TAU_REGISTER_EVENT(variable, event_name);
    TAU_EVENT(variable, value);
    TAU_PROFILE_STMT(statement);
- Heap Memory Tracking:
  - TAU_TRACK_MEMORY();
  - TAU_TRACK_MEMORY_HEADROOM();
  - TAU_SET_INTERRUPT_INTERVAL(seconds);
  - TAU_DISABLE_TRACKING_MEMORY[_HEADROOM]();
  - TAU_ENABLE_TRACKING_MEMORY[_HEADROOM]();



Manual Instrumentation – C/C++ Example

```
#include <TAU.h>
int main(int argc, char **argv)
{
  TAU_START (“big-loop”)

  for(int i = 0; i < N ; i++){
    work(i);
  }

  TAU_STOP (“big-loop”);

}
```

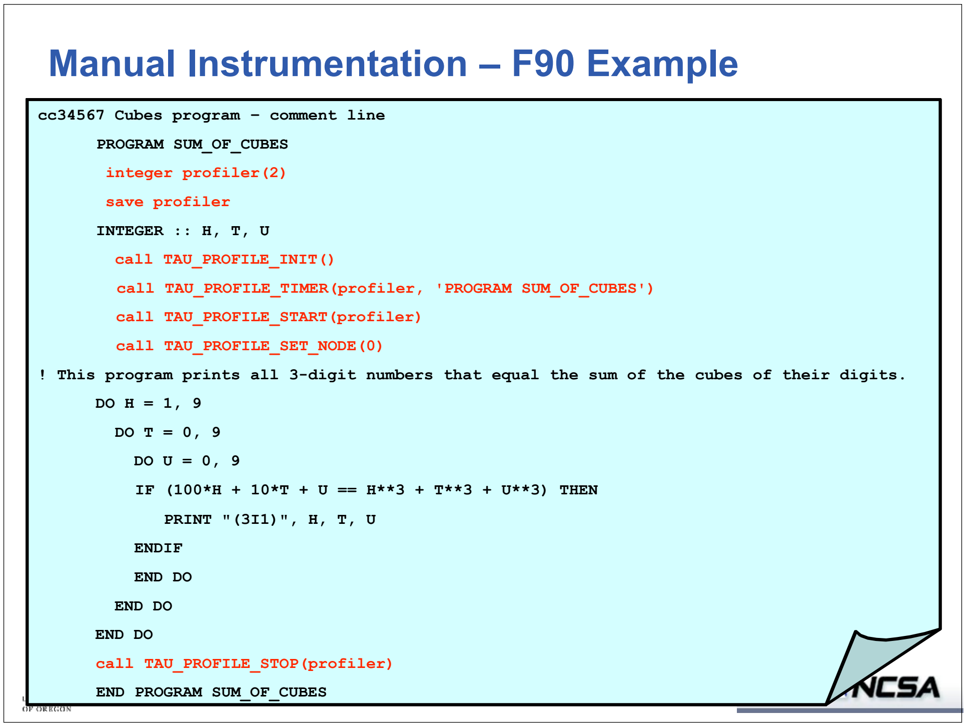


Manual Instrumentation – C++ Example

```
#include <TAU.h>
int main(int argc, char **argv)
{
  TAU_PROFILE(“int main(int, char **)”, “ ”, TAU_DEFAULT);
  TAU_PROFILE_INIT(argc, argv);
  TAU_PROFILE_SET_NODE(0); /* for sequential programs */
  foo();
  return 0;
}
int foo(void)
{
  TAU_PROFILE(“int foo(void)”, “ ”, TAU_DEFAULT); // measures entire foo()
   TAU_PROFILE_TIMER(t, “foo(): for loop”, “[23:45 file.cpp]”, TAU_USER);
  TAU_PROFILE_START(t);
  for(int i = 0; i < N ; i++){
    work(i);
  }
  TAU_PROFILE_STOP(t);
  // other statements in foo …
}
```

121

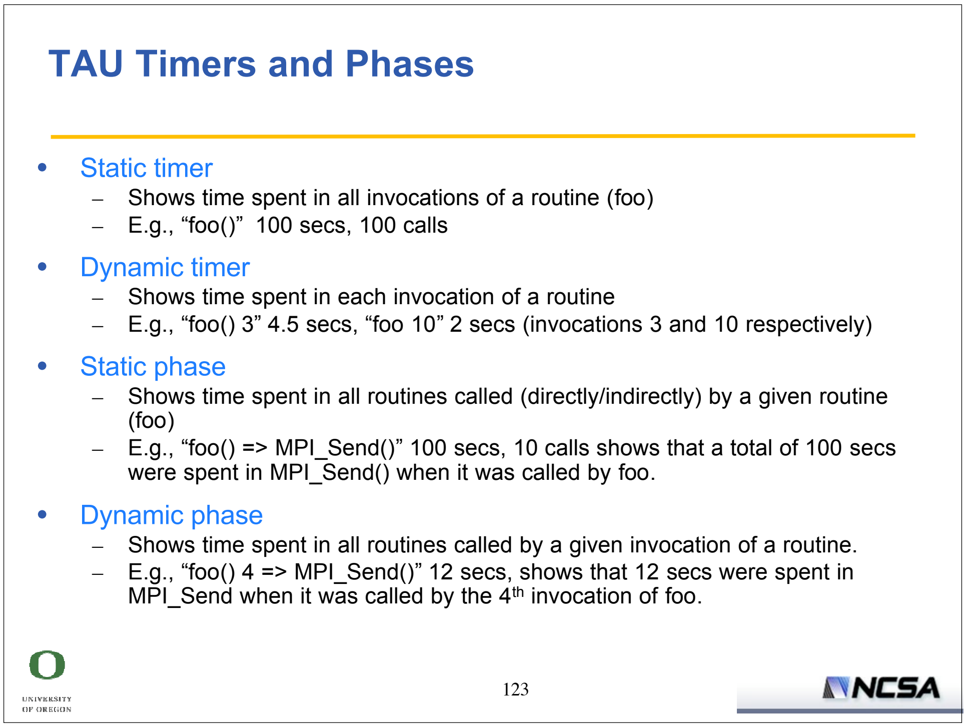
UNIVERSITY
OF OREGON



Manual Instrumentation – F90 Example

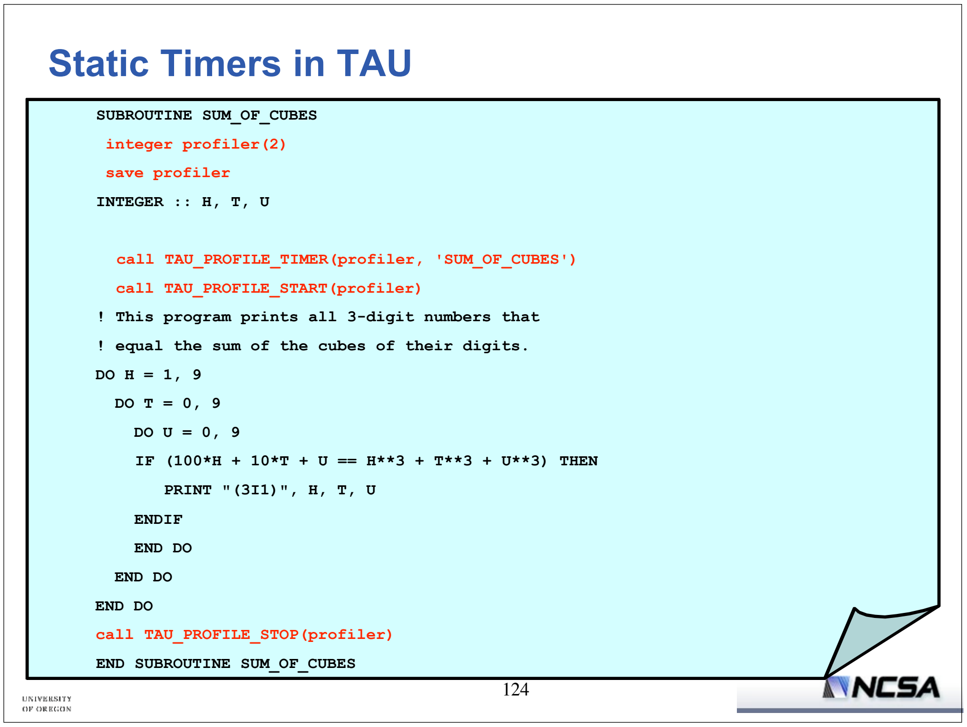
```
cc34567 Cubes program – comment line
      PROGRAM SUM_OF_CUBES
       integer profiler(2)
       save profiler
      INTEGER :: H, T, U
        call TAU_PROFILE_INIT()
        call TAU_PROFILE_TIMER(profiler, 'PROGRAM SUM_OF_CUBES')
        call TAU_PROFILE_START(profiler)
        call TAU_PROFILE_SET_NODE(0)
! This program prints all 3-digit numbers that equal the sum of the cubes of their digits.
      DO H = 1, 9
        DO T = 0, 9
          DO U = 0, 9
          IF (100*H + 10*T + U == H**3 + T**3 + U**3) THEN
             PRINT "(3I1)", H, T, U
          ENDIF
          END DO
        END DO
      END DO
      call TAU_PROFILE_STOP(profiler)
      END PROGRAM SUM_OF_CUBES
```

OF OREGON



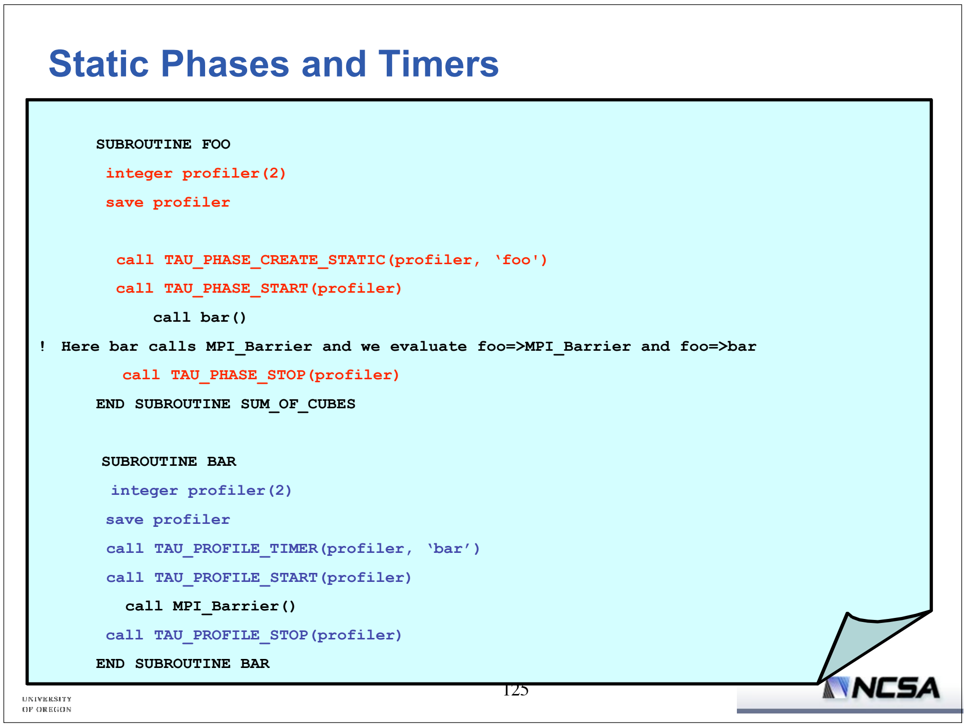
TAU Timers and Phases

- Static timer
  - Shows time spent in all invocations of a routine (foo)
  - E.g., “foo()” 100 secs, 100 calls
- Dynamic timer
  - Shows time spent in each invocation of a routine
  - E.g., “foo() 3” 4.5 secs, “foo 10” 2 secs (invocations 3 and 10 respectively)
- Static phase
  - Shows time spent in all routines called (directly/indirectly) by a given routine (foo)
  - E.g., “foo() => MPI_Send()” 100 secs, 10 calls shows that a total of 100 secs were spent in MPI_Send() when it was called by foo.
- Dynamic phase
  - Shows time spent in all routines called by a given invocation of a routine.
  - E.g., “foo() 4 => MPI_Send()” 12 secs, shows that 12 secs were spent in MPI_Send when it was called by the 4th invocation of foo.



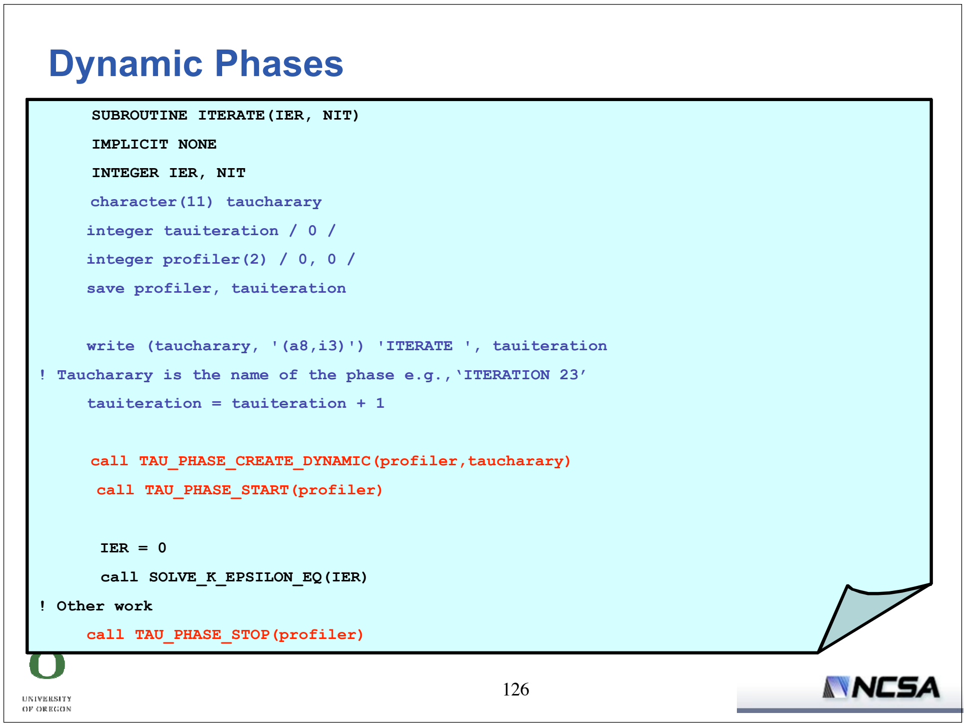
Static Timers in TAU

```
SUBROUTINE SUM_OF_CUBES
 integer profiler(2)
 save profiler
INTEGER :: H, T, U

  call TAU_PROFILE_TIMER(profiler, 'SUM_OF_CUBES')
  call TAU_PROFILE_START(profiler)
! This program prints all 3-digit numbers that
! equal the sum of the cubes of their digits.
DO H = 1, 9
  DO T = 0, 9
    DO U = 0, 9
    IF (100*H + 10*T + U == H**3 + T**3 + U**3) THEN
       PRINT "(3I1)", H, T, U
    ENDIF
    END DO
  END DO
END DO
call TAU_PROFILE_STOP(profiler)
END SUBROUTINE SUM_OF_CUBES
```



Static Phases and Timers

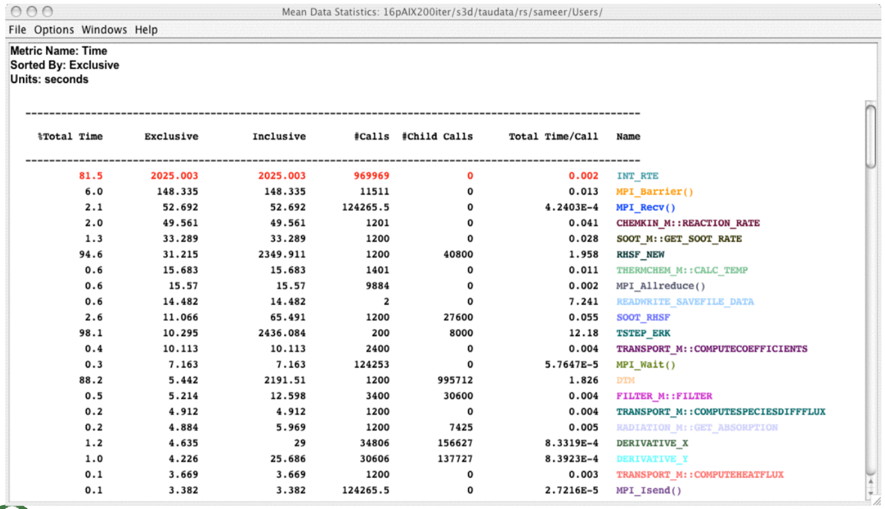
```
      SUBROUTINE FOO
       integer profiler(2)
       save profiler

        call TAU_PHASE_CREATE_STATIC(profiler, ‘foo')
        call TAU_PHASE_START(profiler)
            call bar()
!  Here bar calls MPI_Barrier and we evaluate foo=>MPI_Barrier and foo=>bar
         call TAU_PHASE_STOP(profiler)
      END SUBROUTINE SUM_OF_CUBES

       SUBROUTINE BAR
        integer profiler(2)
       save profiler
       call TAU_PROFILE_TIMER(profiler, ‘bar’)
       call TAU_PROFILE_START(profiler)
         call MPI_Barrier()
       call TAU_PROFILE_STOP(profiler)
      END SUBROUTINE BAR
```



Dynamic Phases

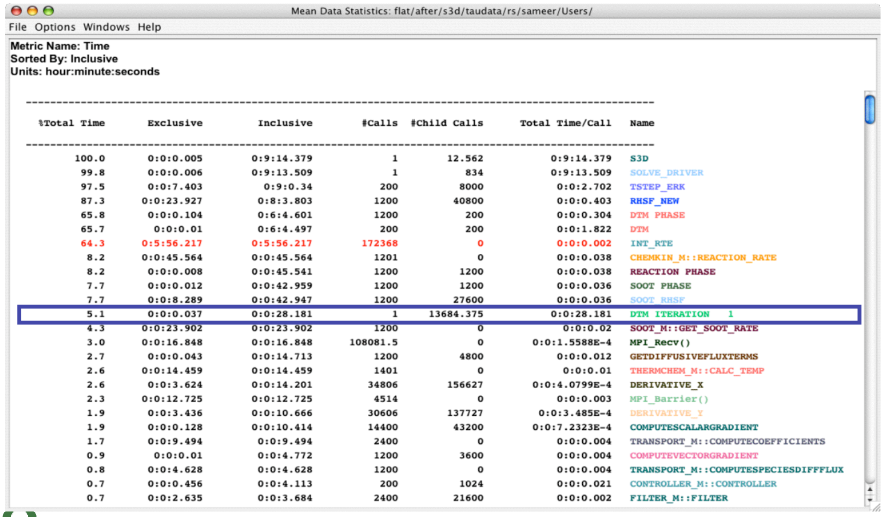
```
      SUBROUTINE ITERATE(IER, NIT)
      IMPLICIT NONE
      INTEGER IER, NIT
      character(11) taucharary
     integer tauiteration / 0 /
     integer profiler(2) / 0, 0 /
     save profiler, tauiteration

     write (taucharary, '(a8,i3)') 'ITERATE ', tauiteration
! Taucharary is the name of the phase e.g.,‘ITERATION 23’
     tauiteration = tauiteration + 1

      call TAU_PHASE_CREATE_DYNAMIC(profiler,taucharary)
       call TAU_PHASE_START(profiler)

       IER = 0
       call SOLVE_K_EPSILON_EQ(IER)
! Other work
     call TAU_PHASE_STOP(profiler)
```

TAU's ParaProf Profile Browser: Static Timers


Mean Data Statistics: 16pAIX200iter/s3d/taudata/rs/sameer/Users/
File Options Windows Help

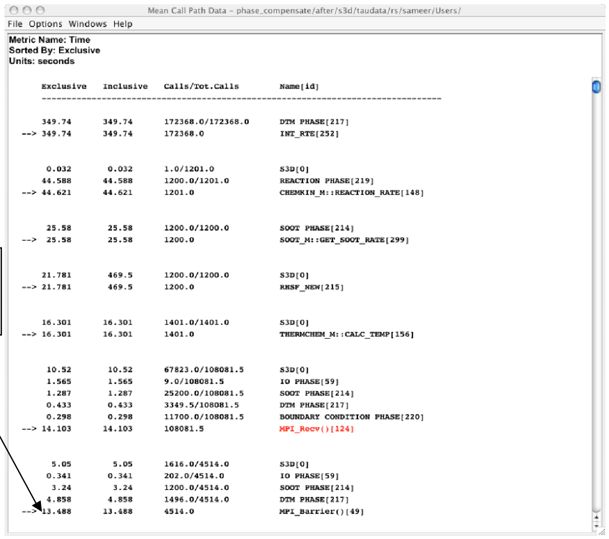
```
Metric Name: Time
Sorted By: Exclusive
Units: seconds

   ---------------------------------------------------------------------------------------------------

     %Total Time      Exclusive       Inclusive      #Calls  #Child Calls     Total Time/Call   Name

   ---------------------------------------------------------------------------------------------------
            81.5       2025.003        2025.003      969969             0               0.002   INT_RTE
             6.0        148.335         148.335       11511             0               0.013   MPI_Barrier()
             2.1         52.692          52.692    124265.5             0           4.2403E-4   MPI_Recv()
             2.0         49.561          49.561        1201             0               0.041   CHEMKIN_M::REACTION_RATE
             1.3         33.289          33.289        1200             0               0.028   SOOT_M::GET_SOOT_RATE
            94.6         31.215        2349.911        1200         40800               1.958   RHSF_NEW
             0.6         15.683          15.683        1401             0               0.011   THERMCHEM_M::CALC_TEMP
             0.6          15.57           15.57        9884             0               0.002   MPI_Allreduce()
             0.6         14.482          14.482           2             0               7.241   READWRITE_SAVEFILE_DATA
             2.6         11.066          65.491        1200         27600               0.055   SOOT_RHSF
            98.1         10.295        2436.084         200          8000               12.18   TSTEP_ERK
             0.4         10.113          10.113        2400             0               0.004   TRANSPORT_M::COMPUTECOEFFICIENTS
             0.3          7.163           7.163      124253             0           5.7647E-5   MPI_Wait()
            88.2          5.442         2191.51        1200        995712               1.826   DTM
             0.5          5.214          12.598        3400         30600               0.004   FILTER_M::FILTER
             0.2          4.912           4.912        1200             0               0.004   TRANSPORT_M::COMPUTESPECIESDIFFFLUX
             0.2          4.884           5.969        1200          7425               0.005   RADIATION_M::GET_ABSORPTION
             1.2          4.635              29       34806        156627           8.3319E-4   DERIVATIVE_X
             1.0          4.226          25.686       30606        137727           8.3923E-4   DERIVATIVE_Y
             0.1          3.669           3.669        1200             0               0.003   TRANSPORT_M::COMPUTEHEATFLUX
             0.1          3.382           3.382    124265.5             0           2.7216E-5   MPI_Isend()
```

Dynamic Timers


Mean Data Statistics: flat/after/s3d/taudata/rs/sameer/Users/
File Options Windows Help

```
Metric Name: Time
Sorted By: Inclusive
Units: hour:minute:seconds

   ---------------------------------------------------------------------------------------------------

     %Total Time      Exclusive        Inclusive      #Calls  #Child Calls     Total Time/Call   Name

   ---------------------------------------------------------------------------------------------------
           100.0      0:0:0.005       0:9:14.379           1        12.562          0:9:14.379   S3D
            99.8      0:0:0.006       0:9:13.509           1           834          0:9:13.509   SOLVE_DRIVER
            97.5      0:0:7.403         0:9:0.34         200          8000           0:0:2.702   TSTEP_ERK
            87.3     0:0:23.927        0:8:3.803        1200         40800           0:0:0.403   RHSF_NEW
            65.8      0:0:0.104        0:6:4.601        1200           200           0:0:0.304   DTM PHASE
            65.7       0:0:0.01        0:6:4.497         200           200           0:0:1.822   DTM
            64.3      0:5:56.217       0:5:56.217      172368             0           0:0:0.002   INT_RTE
             8.2     0:0:45.564       0:0:45.564        1201             0           0:0:0.038   CHEMKIN_M::REACTION_RATE
             8.2      0:0:0.008       0:0:45.541        1200          1200           0:0:0.038   REACTION PHASE
             7.7      0:0:0.012       0:0:42.959        1200          1200           0:0:0.036   SOOT PHASE
             7.7      0:0:8.289       0:0:42.947        1200         27600           0:0:0.036   SOOT_RHSF
             5.1      0:0:0.037       0:0:28.181           1     13684.375          0:0:28.181   DTM ITERATION   1
             4.3     0:0:23.902       0:0:23.902        1200             0            0:0:0.02   SOOT_M::GET_SOOT_RATE
             3.0     0:0:16.848       0:0:16.848    108081.5             0      0:0:1.5588E-4   MPI_Recv()
             2.7      0:0:0.043       0:0:14.713        1200          4800           0:0:0.012   GETDIFFUSIVEFLUXTERMS
             2.6     0:0:14.459       0:0:14.459        1401             0            0:0:0.01   THERMCHEM_M::CALC_TEMP
             2.6      0:0:3.624       0:0:14.201       34806        156627      0:0:4.0799E-4   DERIVATIVE_X
             2.3     0:0:12.725       0:0:12.725        4514             0           0:0:0.003   MPI_Barrier()
             1.9      0:0:3.436       0:0:10.666       30606        137727       0:0:3.485E-4   DERIVATIVE_Y
             1.9      0:0:0.128       0:0:10.414       14400         43200      0:0:7.2323E-4   COMPUTESCALARGRADIENT
             1.7      0:0:9.494        0:0:9.494        2400             0           0:0:0.004   TRANSPORT_M::COMPUTECOEFFICIENTS
             0.9       0:0:0.01        0:0:4.772        1200          3600           0:0:0.004   COMPUTEVECTORGRADIENT
             0.8      0:0:4.628        0:0:4.628        1200             0           0:0:0.004   TRANSPORT_M::COMPUTESPECIESDIFFFLUX
             0.7      0:0:0.456        0:0:4.113         200          1024           0:0:0.021   CONTROLLER_M::CONTROLLER
             0.7      0:0:2.635        0:0:3.684        2400         21600           0:0:0.002   FILTER_M::FILTER
```

Static Phases

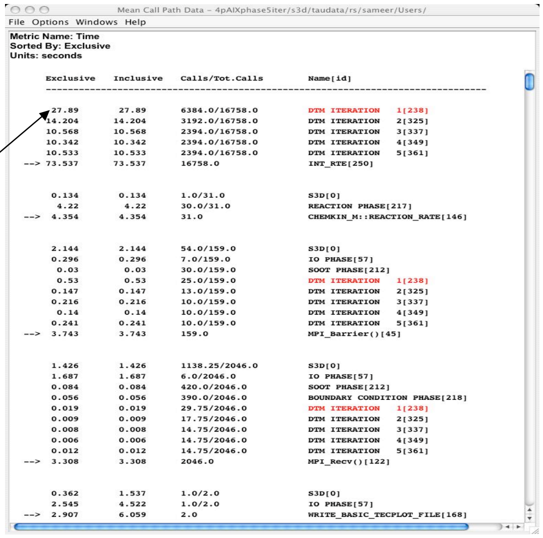


Mean Call Path Data - phase_compensate/after/s3d/taudata/rs/sameer/Users/

```
Metric Name: Time
Sorted By: Exclusive
Units: seconds

      Exclusive   Inclusive   Calls/Tot.Calls        Name[id]
      -------------------------------------------------------------------------

      349.74      349.74      172368.0/172368.0      DTM PHASE[217]
  --> 349.74      349.74      172368.0               INT_RTE[252]


       0.032       0.032      1.0/1201.0             S3D[0]
      44.588      44.588      1200.0/1201.0          REACTION PHASE[219]
  --> 44.621      44.621      1201.0                 CHEMKIN_M::REACTION_RATE[148]


       25.58       25.58      1200.0/1200.0          SOOT PHASE[214]
  -->  25.58       25.58      1200.0                 SOOT_M::GET_SOOT_RATE[299]


      21.781       469.5      1200.0/1200.0          S3D[0]
  --> 21.781       469.5      1200.0                 RHSF_NEW[215]


      16.301      16.301      1401.0/1401.0          S3D[0]
  --> 16.301      16.301      1401.0                 THERMCHEM_M::CALC_TEMP[156]


       10.52       10.52      67823.0/108081.5       S3D[0]
       1.565       1.565      9.0/108081.5           IO PHASE[59]
       1.287       1.287      25200.0/108081.5       SOOT PHASE[214]
       0.433       0.433      3349.5/108081.5        DTM PHASE[217]
       0.298       0.298      11700.0/108081.5       BOUNDARY CONDITION PHASE[220]
  --> 14.103      14.103      108081.5               MPI_Recv()[124]


        5.05        5.05      1616.0/4514.0          S3D[0]
       0.341       0.341      202.0/4514.0           IO PHASE[59]
        3.24        3.24      1200.0/4514.0          SOOT PHASE[214]
       4.858       4.858      1496.0/4514.0          DTM PHASE[217]
  --> 13.488      13.488      4514.0                 MPI_Barrier()[49]
```

MPI_Barrier took 4.85 secs out of 13.48 secs in the DTM Phase

Dynamic Phases

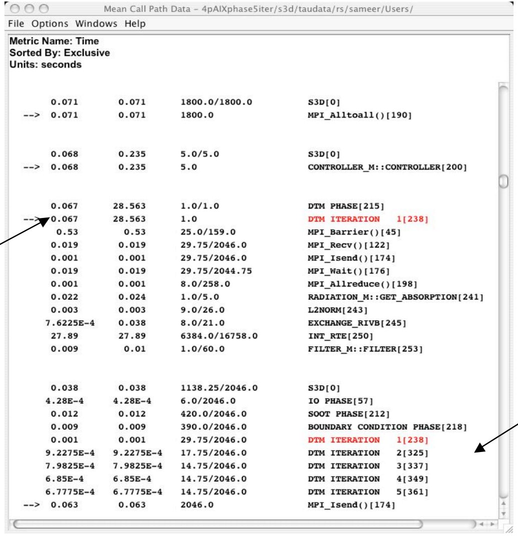


Mean Call Path Data - 4pAIXphase5iter/s3d/taudata/rs/sameer/Users/

```
Metric Name: Time
Sorted By: Exclusive
Units: seconds

      Exclusive   Inclusive   Calls/Tot.Calls        Name[id]
      -------------------------------------------------------------------------

       27.89       27.89      6384.0/16758.0         DTM ITERATION   1[238]
      14.204      14.204      3192.0/16758.0         DTM ITERATION   2[325]
      10.568      10.568      2394.0/16758.0         DTM ITERATION   3[337]
      10.342      10.342      2394.0/16758.0         DTM ITERATION   4[349]
      10.533      10.533      2394.0/16758.0         DTM ITERATION   5[361]
  --> 73.537      73.537      16758.0                INT_RTE[250]


       0.134       0.134      1.0/31.0               S3D[0]
        4.22        4.22      30.0/31.0              REACTION PHASE[217]
  -->  4.354       4.354      31.0                   CHEMKIN_M::REACTION_RATE[146]


       2.144       2.144      54.0/159.0             S3D[0]
       0.296       0.296      7.0/159.0              IO PHASE[57]
        0.03        0.03      30.0/159.0             SOOT PHASE[212]
        0.53        0.53      25.0/159.0             DTM ITERATION   1[238]
       0.147       0.147      13.0/159.0             DTM ITERATION   2[325]
       0.216       0.216      10.0/159.0             DTM ITERATION   3[337]
        0.14        0.14      10.0/159.0             DTM ITERATION   4[349]
       0.241       0.241      10.0/159.0             DTM ITERATION   5[361]
  -->  3.743       3.743      159.0                  MPI_Barrier()[45]


       1.426       1.426      1138.25/2046.0         S3D[0]
       1.687       1.687      6.0/2046.0             IO PHASE[57]
       0.084       0.084      420.0/2046.0           SOOT PHASE[212]
       0.056       0.056      390.0/2046.0           BOUNDARY CONDITION PHASE[218]
       0.019       0.019      29.75/2046.0           DTM ITERATION   1[238]
       0.009       0.009      17.75/2046.0           DTM ITERATION   2[325]
       0.008       0.008      14.75/2046.0           DTM ITERATION   3[337]
       0.006       0.006      14.75/2046.0           DTM ITERATION   4[349]
       0.012       0.012      14.75/2046.0           DTM ITERATION   5[361]
  -->  3.308       3.308      2046.0                 MPI_Recv()[122]


       0.362       1.537      1.0/2.0                S3D[0]
       2.545       4.522      1.0/2.0                IO PHASE[57]
  -->  2.907       6.059      2.0                    WRITE_BASIC_TECPLOT_FILE[168]
```

The first iteration was expensive for INT_RTE. It took 27.89 secs. Other iterations took less time – 14.2, 10.5, 10.3, 10.5 seconds

Dynamic Phases


Mean Call Path Data – 4pAIXphase5iter/s3d/taudata/rs/sameer/Users/
File Options Windows Help

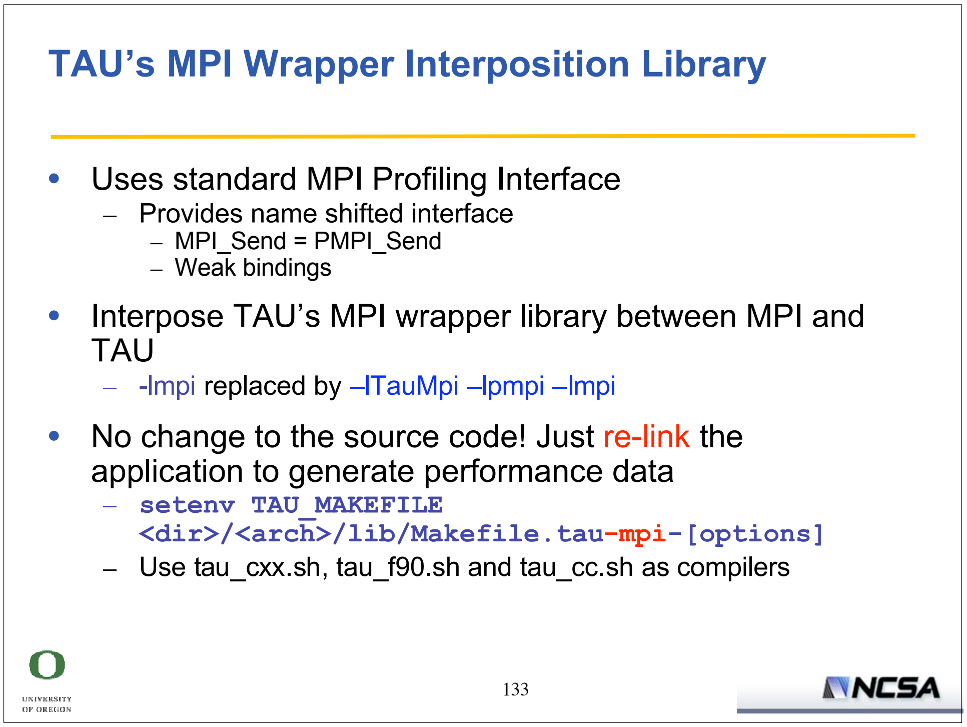
```
Metric Name: Time
Sorted By: Exclusive
Units: seconds


      0.071        0.071       1800.0/1800.0          S3D[0]
 -->  0.071        0.071       1800.0                 MPI_Alltoall()[190]


      0.068        0.235       5.0/5.0                S3D[0]
 -->  0.068        0.235       5.0                    CONTROLLER_M::CONTROLLER[200]


      0.067       28.563       1.0/1.0                DTM PHASE[215]
 -->  0.067       28.563       1.0                    DTM ITERATION   1[238]
       0.53         0.53       25.0/159.0             MPI_Barrier()[45]
      0.019        0.019       29.75/2046.0           MPI_Recv()[122]
      0.001        0.001       29.75/2046.0           MPI_Isend()[174]
      0.019        0.019       29.75/2044.75          MPI_Wait()[176]
      0.001        0.001       8.0/258.0              MPI_Allreduce()[198]
      0.022        0.024       1.0/5.0                RADIATION_M::GET_ABSORPTION[241]
      0.003        0.003       9.0/26.0               L2NORM[243]
    7.6225E-4      0.038       8.0/21.0               EXCHANGE_RIVB[245]
      27.89        27.89       6384.0/16758.0         INT_RTE[250]
      0.009         0.01       1.0/60.0               FILTER_M::FILTER[253]


      0.038        0.038       1138.25/2046.0         S3D[0]
     4.28E-4      4.28E-4      6.0/2046.0             IO PHASE[57]
      0.012        0.012       420.0/2046.0           SOOT PHASE[212]
      0.009        0.009       390.0/2046.0           BOUNDARY CONDITION PHASE[218]
      0.001        0.001       29.75/2046.0           DTM ITERATION   1[238]
    9.2275E-4    9.2275E-4     17.75/2046.0           DTM ITERATION   2[325]
    7.9825E-4    7.9825E-4     14.75/2046.0           DTM ITERATION   3[337]
     6.85E-4      6.85E-4      14.75/2046.0           DTM ITERATION   4[349]
    6.7775E-4    6.7775E-4     14.75/2046.0           DTM ITERATION   5[361]
 -->  0.063        0.063       2046.0                 MPI_Isend()[174]
```

Time spent in MPI_Barrier, MPI_Recv,… in DTM ITERATION 1

Breakdown of time spent in MPI_Isend based on its static and dynamic parent phases

Using TAU – A tutorial

- Configuration
- Instrumentation
  - Manual
  - MPI – Wrapper interposition library
  - PDT- Source rewriting for C,C++, F77/90/95
  - OpenMP – Directive rewriting
  - Component based instrumentation – Proxy components
  - Binary Instrumentation
    - DyninstAPI – Runtime Instrumentation/Rewriting binary
    - Java – Runtime instrumentation
    - Python – Runtime instrumentation
- Measurement
- Performance Analysis

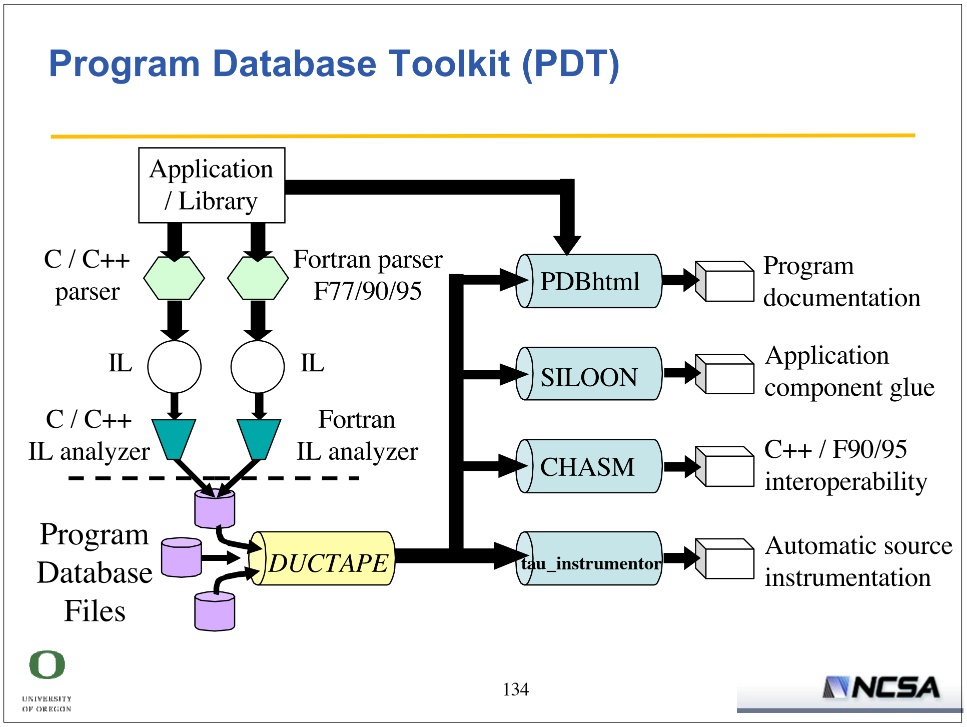


TAU's MPI Wrapper Interposition Library

- Uses standard MPI Profiling Interface
  - Provides name shifted interface
    - MPI_Send = PMPI_Send
    - Weak bindings
- Interpose TAU's MPI wrapper library between MPI and TAU
  - -lmpi replaced by –lTauMpi –lpmpi –lmpi
- No change to the source code! Just re-link the application to generate performance data
  - `setenv TAU_MAKEFILE <dir>/<arch>/lib/Makefile.tau-mpi-[options]`
  - Use tau_cxx.sh, tau_f90.sh and tau_cc.sh as compilers

UNIVERSITY OF OREGON

NCSA



Program Database Toolkit (PDT)

Application / Library

C / C++ parser

Fortran parser F77/90/95

IL

IL

C / C++ IL analyzer

Fortran IL analyzer

Program Database Files

DUCTAPE

PDBhtml

Program documentation

SILOON

Application component glue

CHASM

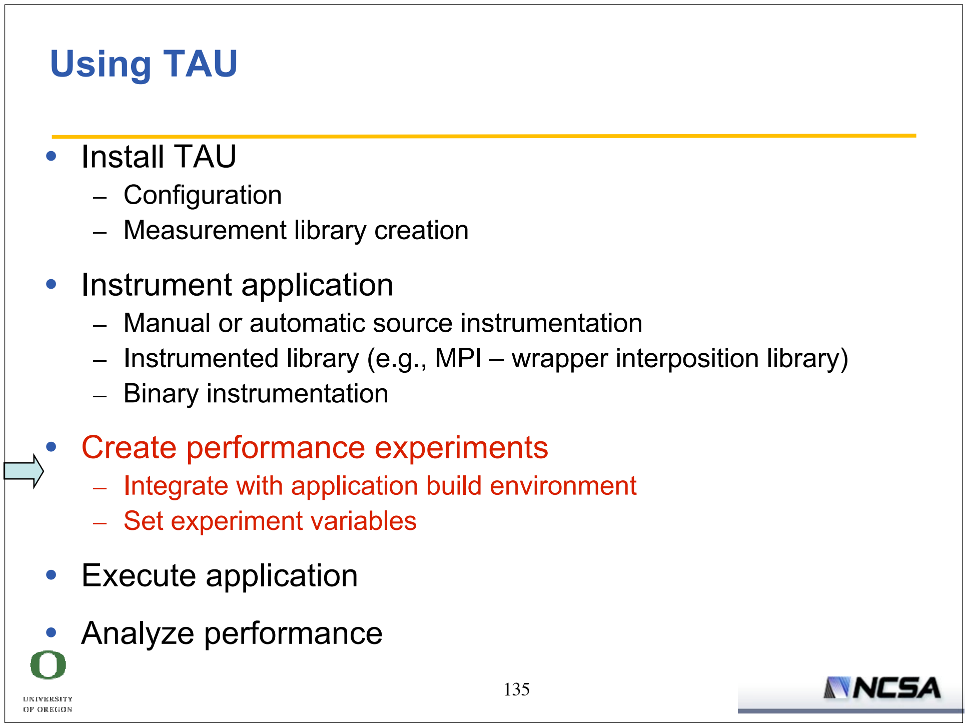
C++ / F90/95 interoperability

tau_instrumentor

Automatic source instrumentation

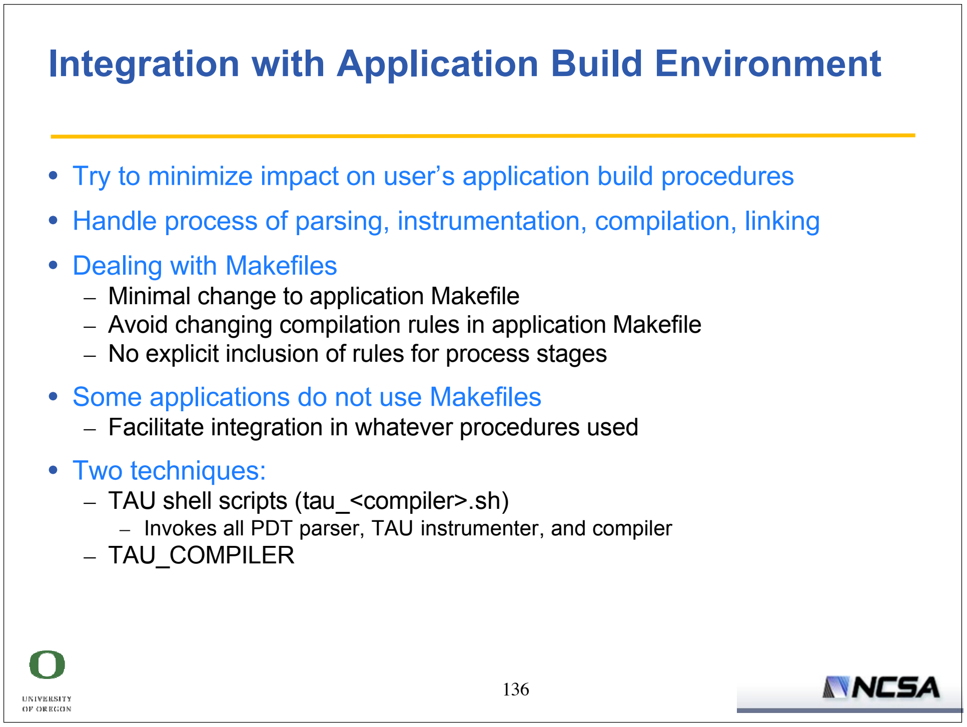
UNIVERSITY OF OREGON

NCSA



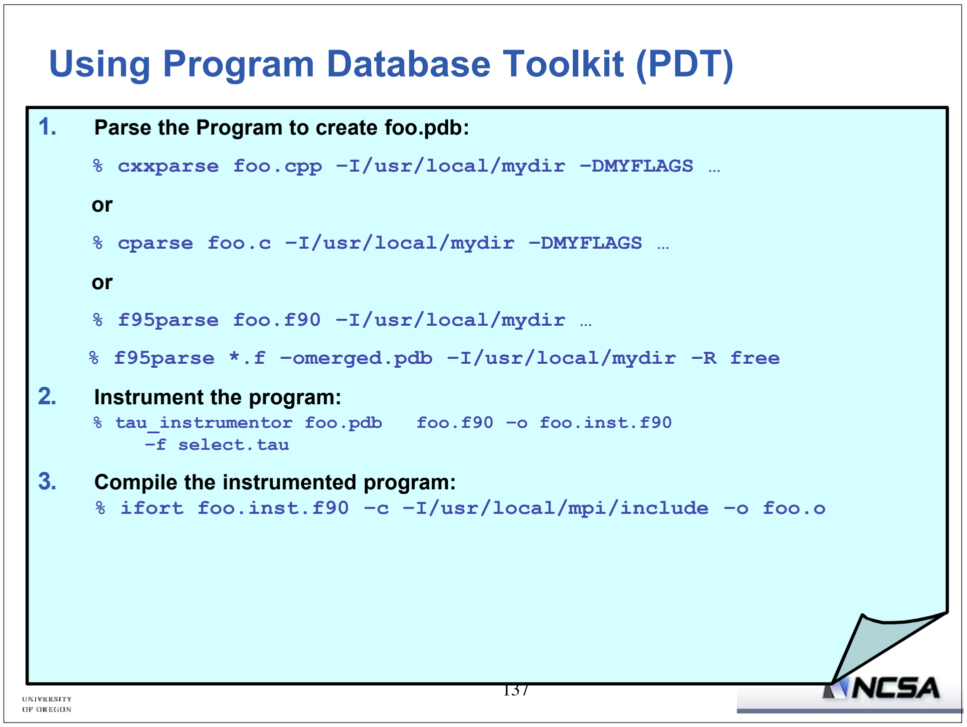
Using TAU

- Install TAU
  - Configuration
  - Measurement library creation
- Instrument application
  - Manual or automatic source instrumentation
  - Instrumented library (e.g., MPI – wrapper interposition library)
  - Binary instrumentation
- Create performance experiments
  - Integrate with application build environment
  - Set experiment variables
- Execute application
- Analyze performance



Integration with Application Build Environment

- Try to minimize impact on user’s application build procedures
- Handle process of parsing, instrumentation, compilation, linking
- Dealing with Makefiles
  - Minimal change to application Makefile
  - Avoid changing compilation rules in application Makefile
  - No explicit inclusion of rules for process stages
- Some applications do not use Makefiles
  - Facilitate integration in whatever procedures used
- Two techniques:
  - TAU shell scripts (tau_<compiler>.sh)
    - Invokes all PDT parser, TAU instrumenter, and compiler
  - TAU_COMPILER



Using Program Database Toolkit (PDT)

1. Parse the Program to create foo.pdb:

```
% cxxparse foo.cpp –I/usr/local/mydir –DMYFLAGS …
```

or

```
% cparse foo.c –I/usr/local/mydir –DMYFLAGS …
```

or

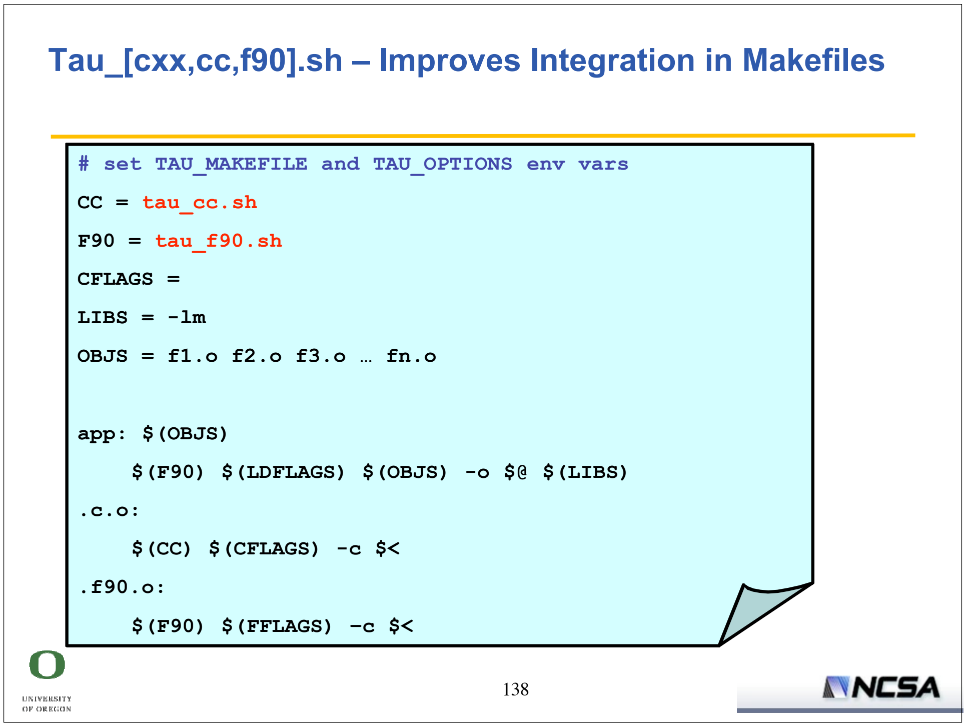
```
% f95parse foo.f90 –I/usr/local/mydir …
% f95parse *.f –omerged.pdb –I/usr/local/mydir –R free
```

2. Instrument the program:

```
% tau_instrumentor foo.pdb   foo.f90 –o foo.inst.f90
    –f select.tau
```

3. Compile the instrumented program:

```
% ifort foo.inst.f90 –c –I/usr/local/mpi/include –o foo.o
```



Tau_[cxx,cc,f90].sh – Improves Integration in Makefiles

```
# set TAU_MAKEFILE and TAU_OPTIONS env vars
CC = tau_cc.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
    $(F90) $(LDFLAGS) $(OBJS) -o $@ $(LIBS)
.c.o:
    $(CC) $(CFLAGS) -c $<
.f90.o:
    $(F90) $(FFLAGS) -c $<
```

AutoInstrumentation using TAU_COMPILER

- $(TAU_COMPILER) stub Makefile variable
- Invokes PDT parser, TAU instrumentor, compiler through `tau_compiler.sh` shell script
- Requires minimal changes to application Makefile
  - Compilation rules are not changed
  - User adds $(TAU_COMPILER) before compiler name
    - F90=mpxlf90
Changes to
F90= $(TAU_COMPILER) mpxlf90
- Passes options from TAU stub Makefile to the four compilation stages
- Use tau_cxx.sh, tau_cc.sh, tau_f90.sh scripts OR $(TAU_COMPILER)
- Uses original compilation command if an error occurs

Automatic Instrumentation

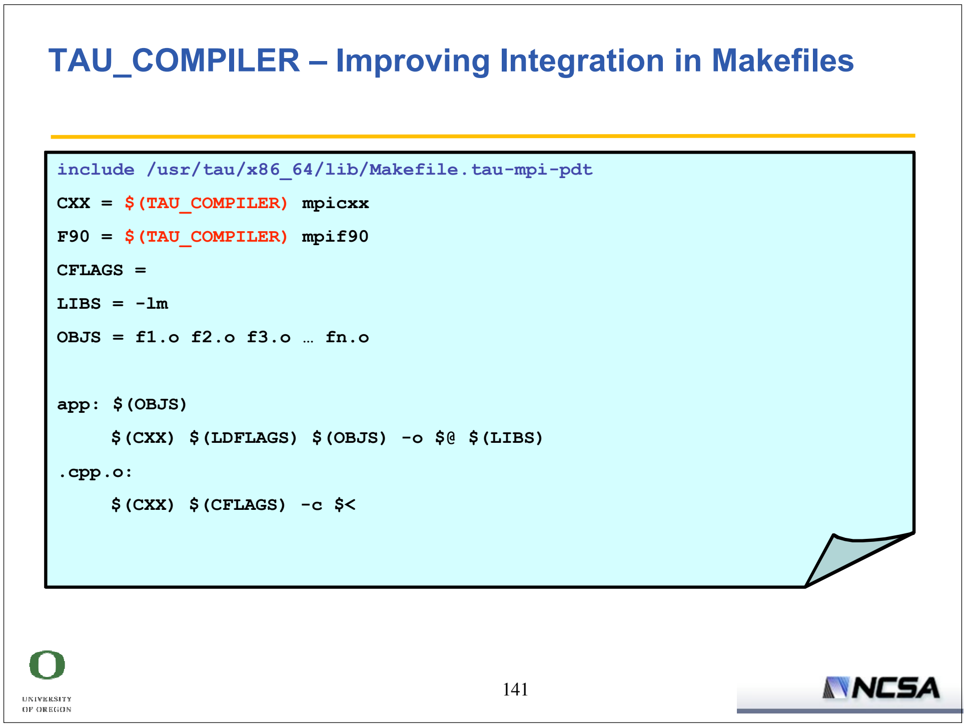
- We now provide compiler wrapper scripts
  - Simply replace `mpxlf90` with `tau_f90.sh`
  - Automatically instruments Fortran source code, links with TAU MPI Wrapper libraries.
- Use `tau_cc.sh` and `tau_cxx.sh` for C/C++

Before

```
CXX = mpCC
F90 = mpxlf90_r
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

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

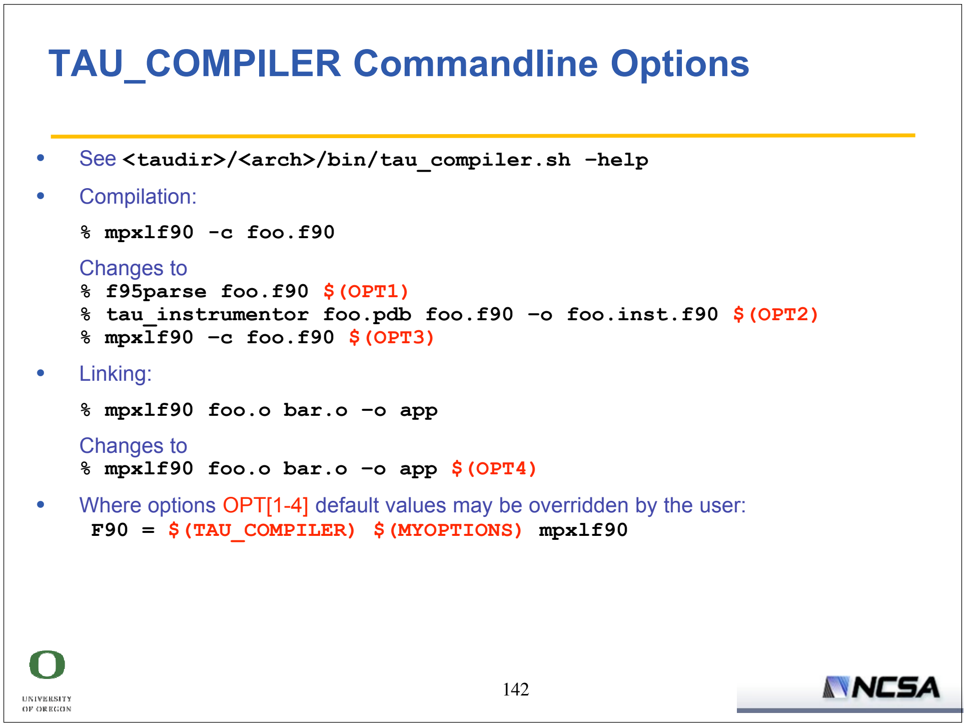
After

```
CXX = tau_cxx.sh
F90 = tau_f90.sh
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

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



TAU_COMPILER – Improving Integration in Makefiles

```
include /usr/tau/x86_64/lib/Makefile.tau-mpi-pdt
CXX = $(TAU_COMPILER) mpicxx
F90 = $(TAU_COMPILER) mpif90
CFLAGS =
LIBS = -lm
OBJS = f1.o f2.o f3.o … fn.o

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



TAU_COMPILER Commandline Options

- See `<taudir>/<arch>/bin/tau_compiler.sh -help`
- Compilation:

  ```
  % mpxlf90 -c foo.f90
  ```

  Changes to

  ```
  % f95parse foo.f90 $(OPT1)
  % tau_instrumentor foo.pdb foo.f90 –o foo.inst.f90 $(OPT2)
  % mpxlf90 –c foo.f90 $(OPT3)
  ```

- Linking:

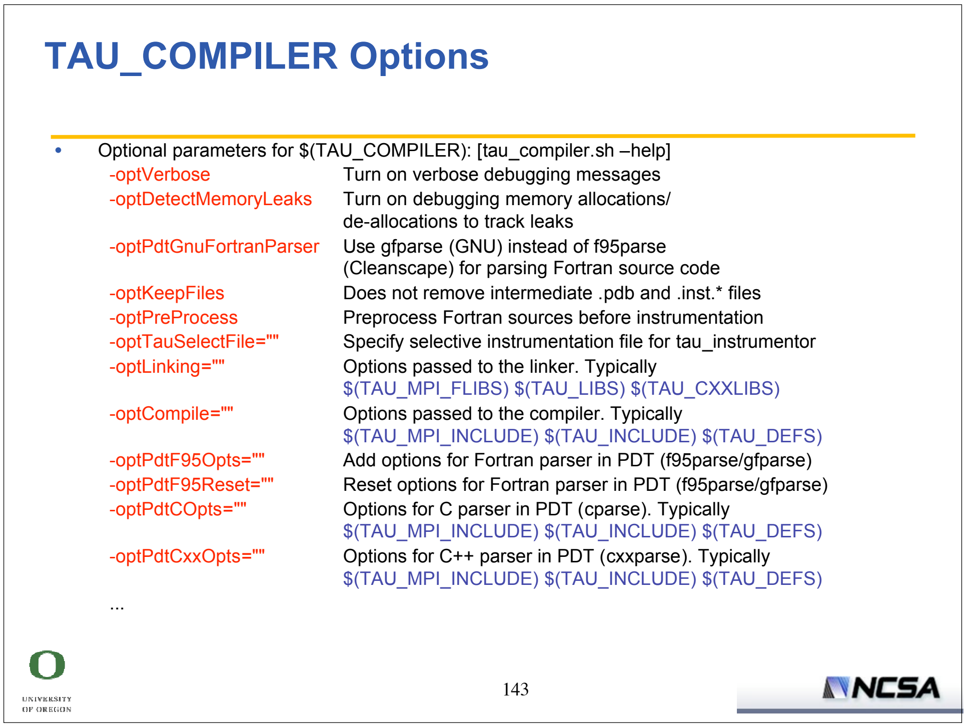
  ```
  % mpxlf90 foo.o bar.o –o app
  ```

  Changes to

  ```
  % mpxlf90 foo.o bar.o –o app $(OPT4)
  ```

- Where options OPT[1-4] default values may be overridden by the user:

  ```
  F90 = $(TAU_COMPILER) $(MYOPTIONS) mpxlf90
  ```

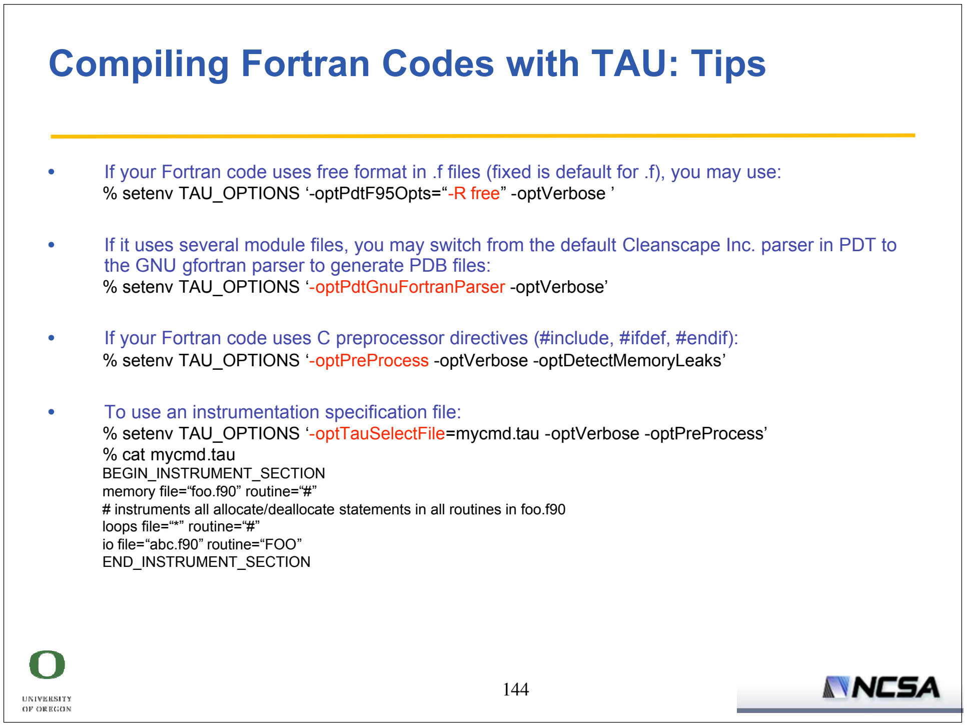


TAU_COMPILER Options

- Optional parameters for $(TAU_COMPILER): [tau_compiler.sh –help]

-optVerbose	Turn on verbose debugging messages
-optDetectMemoryLeaks	Turn on debugging memory allocations/ de-allocations to track leaks
-optPdtGnuFortranParser	Use gfparse (GNU) instead of f95parse (Cleanscape) for parsing Fortran source code
-optKeepFiles	Does not remove intermediate .pdb and .inst.* files
-optPreProcess	Preprocess Fortran sources before instrumentation
-optTauSelectFile=""	Specify selective instrumentation file for tau_instrumentor
-optLinking=""	Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
-optCompile=""	Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtF95Opts=""	Add options for Fortran parser in PDT (f95parse/gfparse)
-optPdtF95Reset=""	Reset options for Fortran parser in PDT (f95parse/gfparse)
-optPdtCOpts=""	Options for C parser in PDT (cparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
-optPdtCxxOpts=""	Options for C++ parser in PDT (cxxparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)

...



Compiling Fortran Codes with TAU: Tips

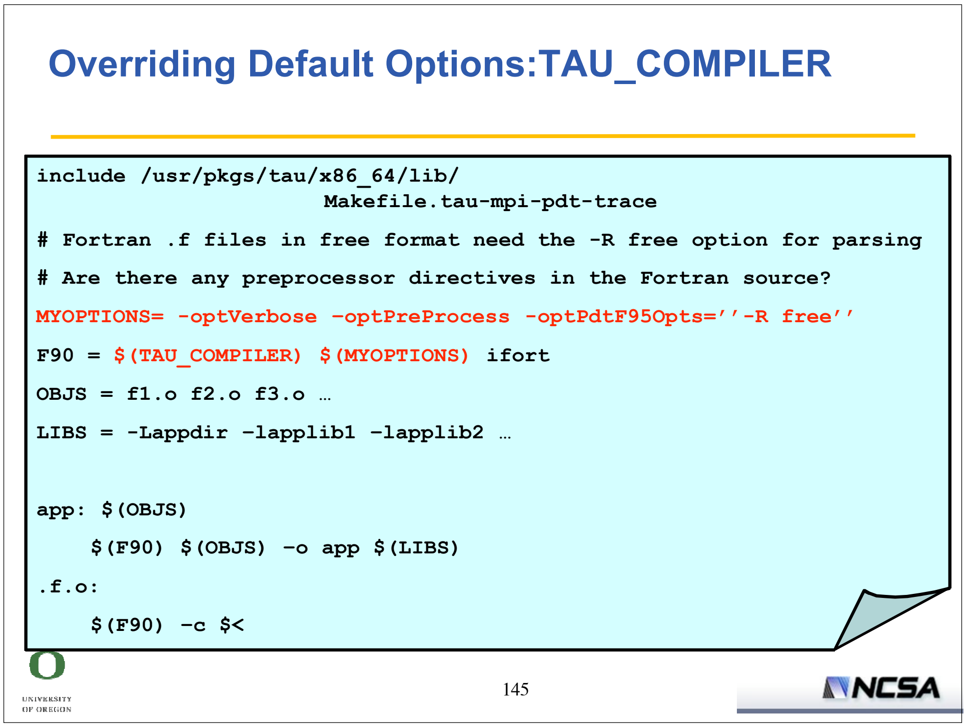
- If your Fortran code uses free format in .f files (fixed is default for .f), you may use:
  % setenv TAU_OPTIONS '-optPdtF95Opts="-R free" -optVerbose '

- If it uses several module files, you may switch from the default Cleanscape Inc. parser in PDT to the GNU gfortran parser to generate PDB files:
  % setenv TAU_OPTIONS '-optPdtGnuFortranParser -optVerbose'

- If your Fortran code uses C preprocessor directives (#include, #ifdef, #endif):
  % setenv TAU_OPTIONS '-optPreProcess -optVerbose -optDetectMemoryLeaks'

- To use an instrumentation specification file:

```
% setenv TAU_OPTIONS '-optTauSelectFile=mycmd.tau -optVerbose -optPreProcess'
% cat mycmd.tau
BEGIN_INSTRUMENT_SECTION
memory file="foo.f90" routine="#"
# instruments all allocate/deallocate statements in all routines in foo.f90
loops file="*" routine="#"
io file="abc.f90" routine="FOO"
END_INSTRUMENT_SECTION
```

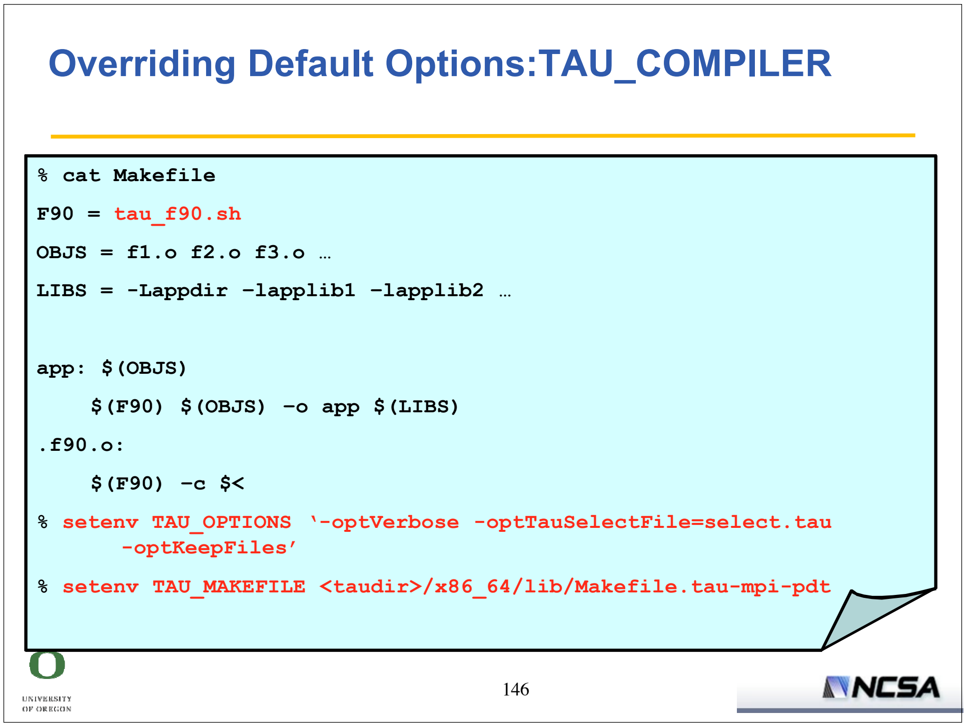


Overriding Default Options:TAU_COMPILER

```
include /usr/pkgs/tau/x86_64/lib/
                      Makefile.tau-mpi-pdt-trace
# Fortran .f files in free format need the -R free option for parsing
# Are there any preprocessor directives in the Fortran source?
MYOPTIONS= -optVerbose -optPreProcess -optPdtF95Opts=’’-R free’’
F90 = $(TAU_COMPILER) $(MYOPTIONS) ifort
OBJS = f1.o f2.o f3.o …
LIBS = -Lappdir -lapplib1 -lapplib2 …

app: $(OBJS)
    $(F90) $(OBJS) -o app $(LIBS)
.f.o:
    $(F90) -c $<
```

UNIVERSITY OF OREGON

NCSA

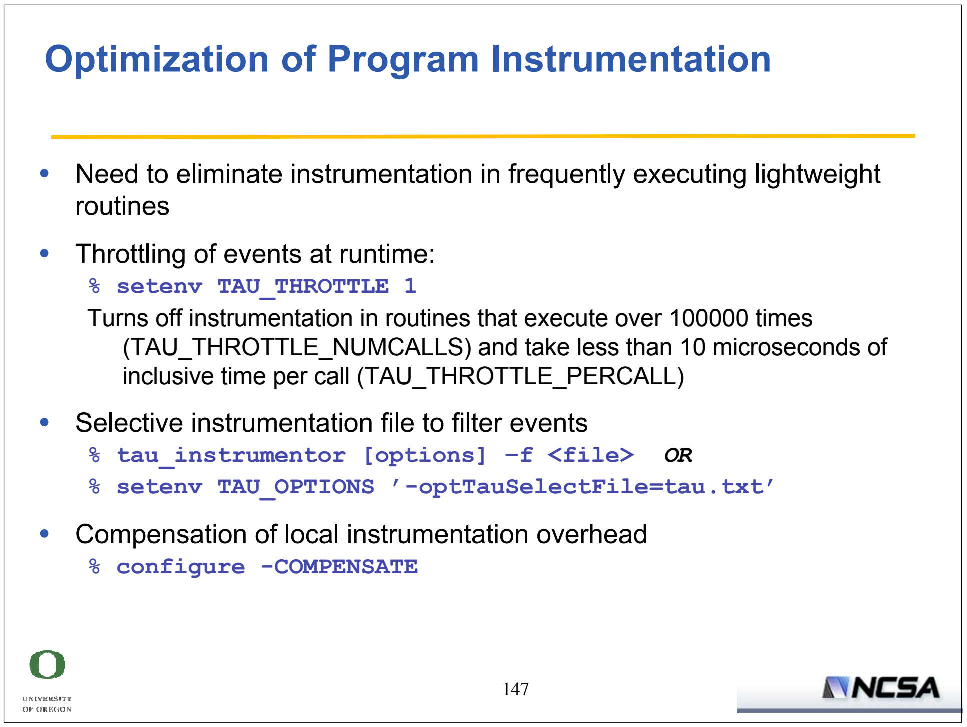


Overriding Default Options:TAU_COMPILER

```
% cat Makefile
F90 = tau_f90.sh
OBJS = f1.o f2.o f3.o …
LIBS = -Lappdir -lapplib1 -lapplib2 …

app: $(OBJS)
    $(F90) $(OBJS) -o app $(LIBS)
.f90.o:
    $(F90) -c $<
% setenv TAU_OPTIONS ‘-optVerbose -optTauSelectFile=select.tau
      -optKeepFiles’
% setenv TAU_MAKEFILE <taudir>/x86_64/lib/Makefile.tau-mpi-pdt
```

UNIVERSITY OF OREGON

NCSA



Optimization of Program Instrumentation

- Need to eliminate instrumentation in frequently executing lightweight routines
- Throttling of events at runtime:

```
% setenv TAU_THROTTLE 1
```

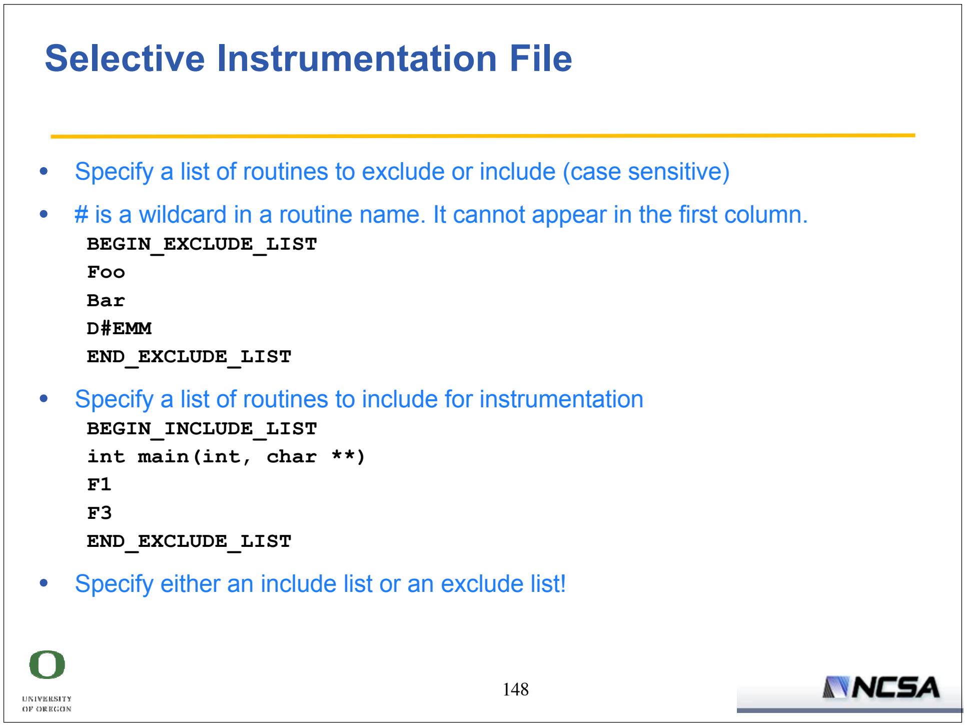
Turns off instrumentation in routines that execute over 100000 times (TAU_THROTTLE_NUMCALLS) and take less than 10 microseconds of inclusive time per call (TAU_THROTTLE_PERCALL)

- Selective instrumentation file to filter events

```
% tau_instrumentor [options] –f <file>   OR
% setenv TAU_OPTIONS '-optTauSelectFile=tau.txt'
```

- Compensation of local instrumentation overhead

```
% configure -COMPENSATE
```



Selective Instrumentation File

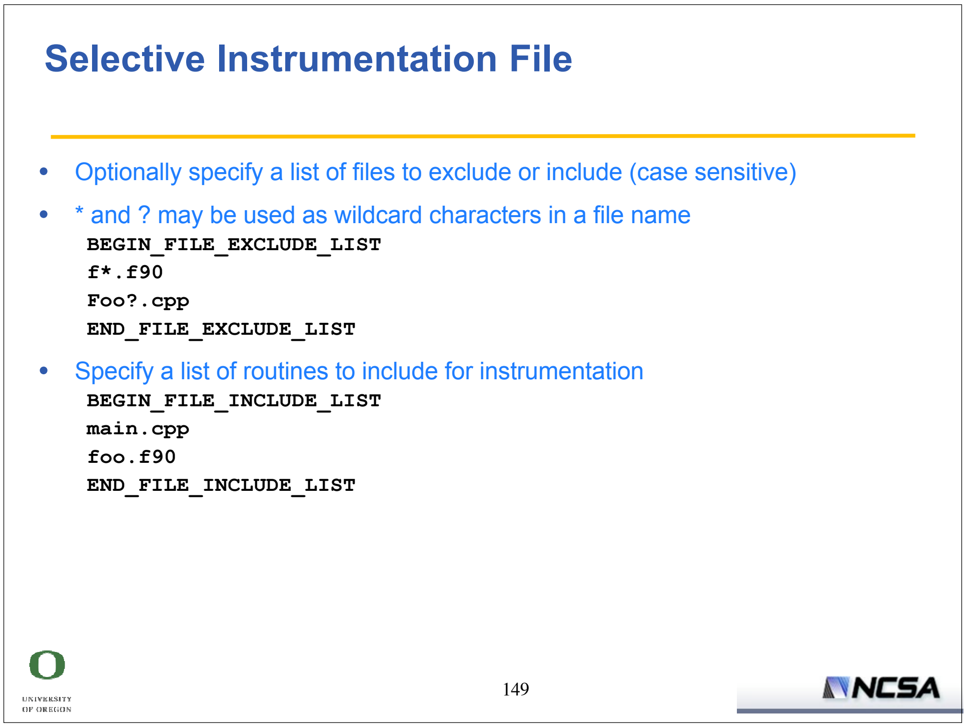
- Specify a list of routines to exclude or include (case sensitive)
- # is a wildcard in a routine name. It cannot appear in the first column.

```
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_EXCLUDE_LIST
```

- Specify either an include list or an exclude list!



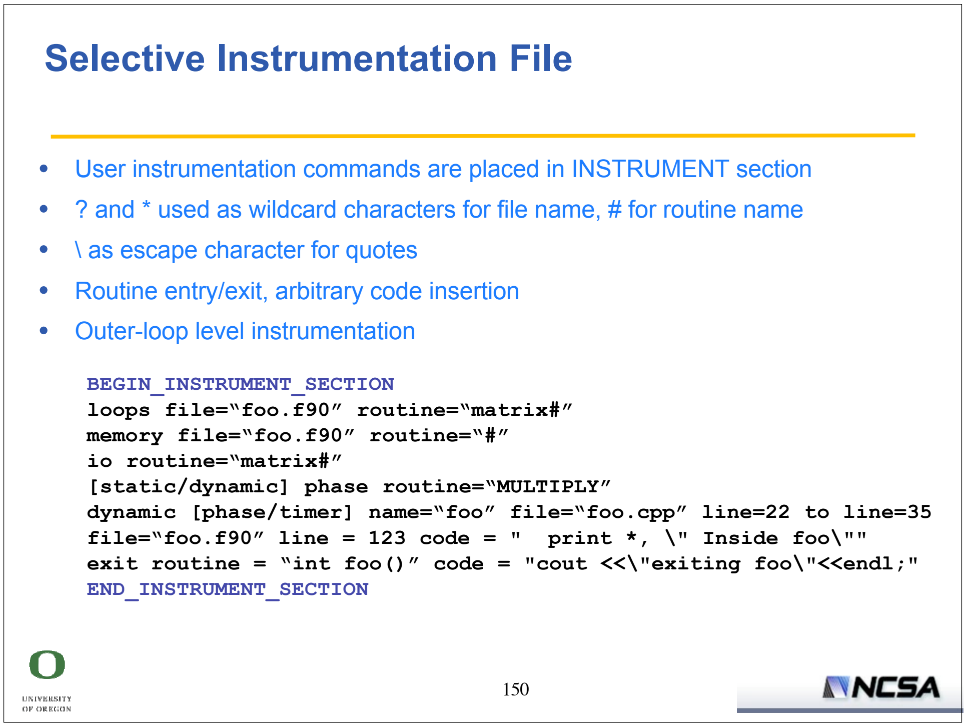
Selective Instrumentation File

- 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

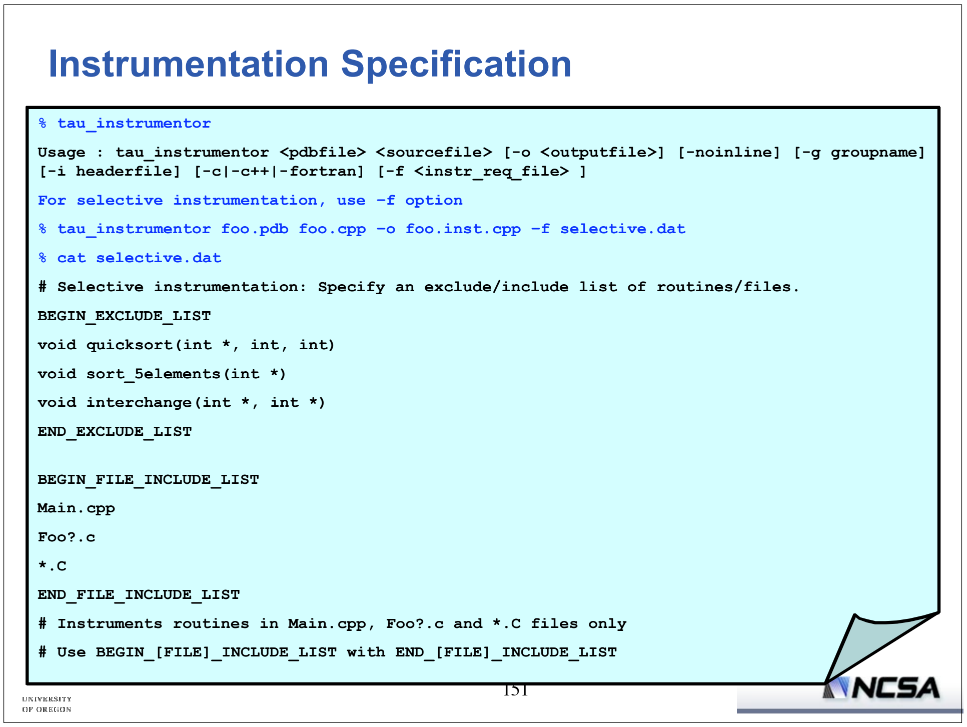
```
BEGIN_FILE_INCLUDE_LIST
main.cpp
foo.f90
END_FILE_INCLUDE_LIST
```



Selective Instrumentation File

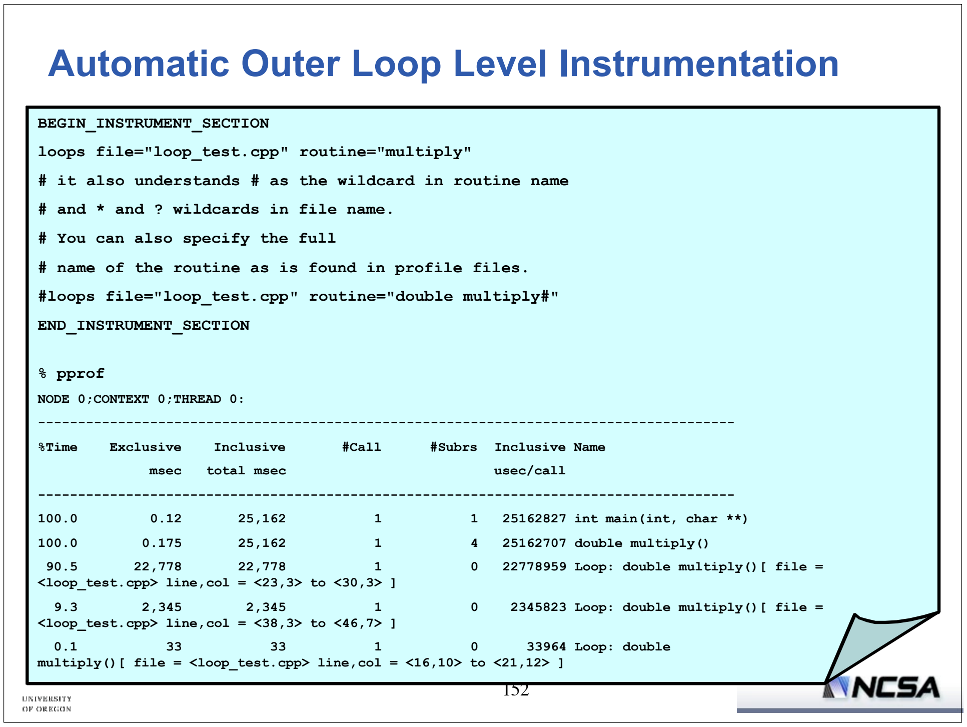
- User instrumentation commands are placed in INSTRUMENT section
- ? and * used as wildcard characters for file name, # for routine name
- \ as escape character for quotes
- Routine entry/exit, arbitrary code insertion
- Outer-loop level instrumentation

```
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
```



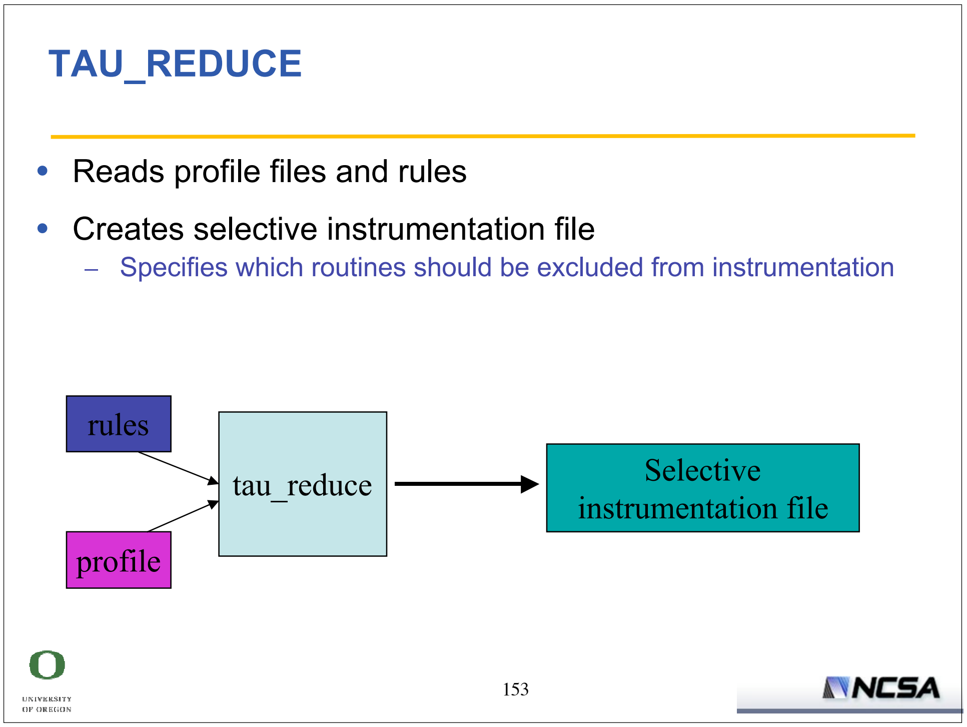
Instrumentation Specification

```
% 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
```



Automatic Outer Loop Level Instrumentation

```
BEGIN_INSTRUMENT_SECTION
loops file="loop_test.cpp" routine="multiply"
# it also understands # as the wildcard in routine name
# and * and ? wildcards in file name.
# You can also specify the full
# name of the routine as is found in profile files.
#loops file="loop_test.cpp" routine="double multiply#"
END_INSTRUMENT_SECTION

% pprof
NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
100.0         0.12       25,162           1           1   25162827 int main(int, char **)
100.0        0.175       25,162           1           4   25162707 double multiply()
 90.5       22,778       22,778           1           0   22778959 Loop: double multiply()[ file =
<loop_test.cpp> line,col = <23,3> to <30,3> ]
  9.3        2,345        2,345           1           0    2345823 Loop: double multiply()[ file =
<loop_test.cpp> line,col = <38,3> to <46,7> ]
  0.1           33           33           1           0      33964 Loop: double
multiply()[ file = <loop_test.cpp> line,col = <16,10> to <21,12> ]
```



TAU_REDUCE

- Reads profile files and rules
- Creates selective instrumentation file
  - Specifies which routines should be excluded from instrumentation

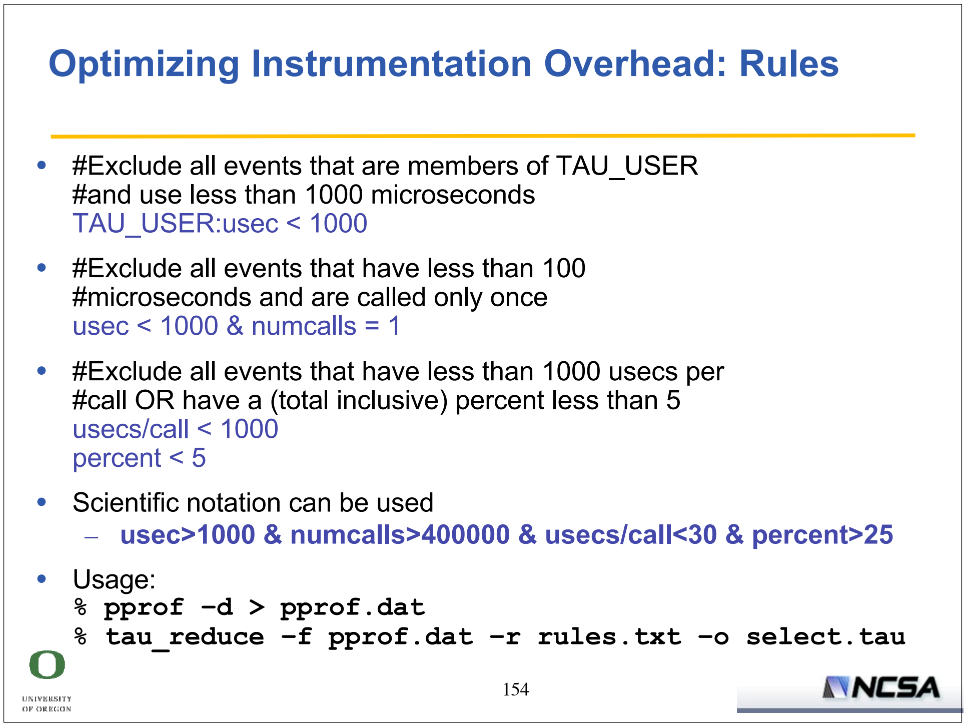
rules

profile

tau_reduce

Selective instrumentation file

153

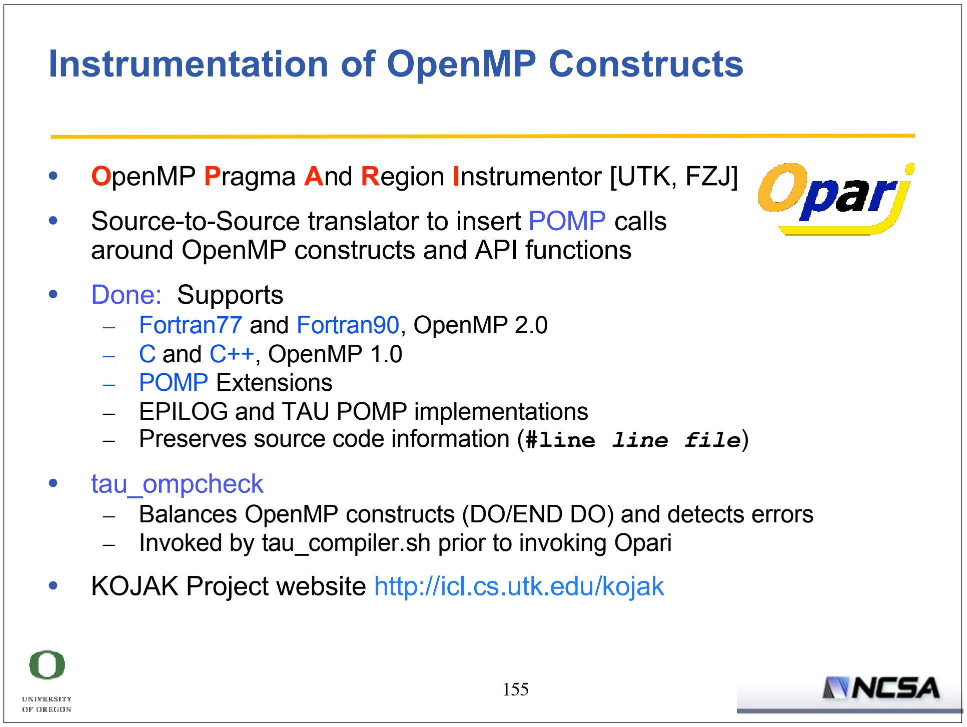


Optimizing Instrumentation Overhead: Rules

- #Exclude all events that are members of TAU_USER
  #and use less than 1000 microseconds
  TAU_USER:usec < 1000
- #Exclude all events that have less than 100
  #microseconds and are called only once
  usec < 1000 & numcalls = 1
- #Exclude all events that have less than 1000 usecs per
  #call OR have a (total inclusive) percent less than 5
  usecs/call < 1000
  percent < 5
- Scientific notation can be used
  - **usec>1000 & numcalls>400000 & usecs/call<30 & percent>25**
- Usage:

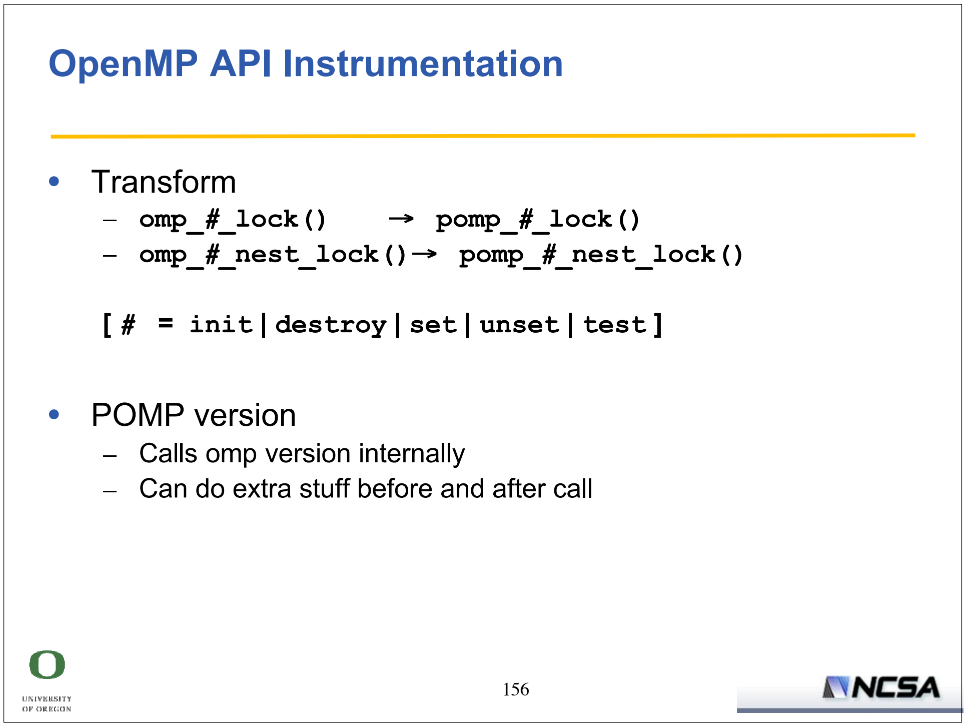
```
% pprof -d > pprof.dat
% tau_reduce -f pprof.dat -r rules.txt -o select.tau
```

154



Instrumentation of OpenMP Constructs

- **O**penMP **P**ragma **A**nd **R**egion **I**nstrumentor [UTK, FZJ]
- Source-to-Source translator to insert POMP calls around OpenMP constructs and API functions
- Done: Supports
  - Fortran77 and Fortran90, OpenMP 2.0
  - C and C++, OpenMP 1.0
  - POMP Extensions
  - EPILOG and TAU POMP implementations
  - Preserves source code information (`#line` ***`line file`***)
- tau_ompcheck
  - Balances OpenMP constructs (DO/END DO) and detects errors
  - Invoked by tau_compiler.sh prior to invoking Opari
- KOJAK Project website http://icl.cs.utk.edu/kojak

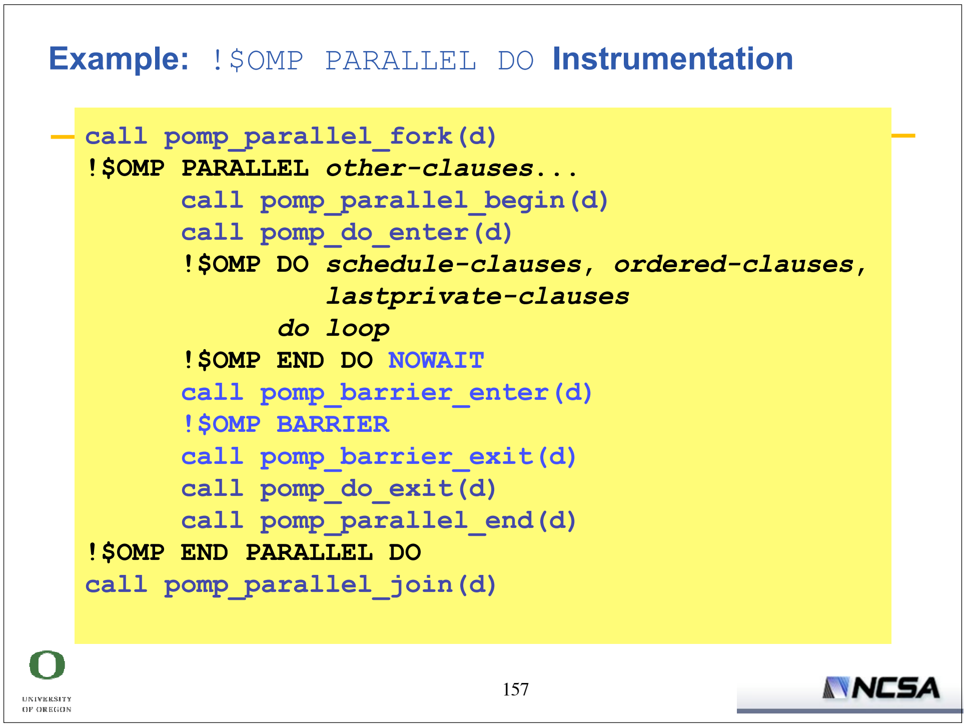


OpenMP API Instrumentation

- Transform
  - **`omp_#_lock()`** → **`pomp_#_lock()`**
  - **`omp_#_nest_lock()`** → **`pomp_#_nest_lock()`**

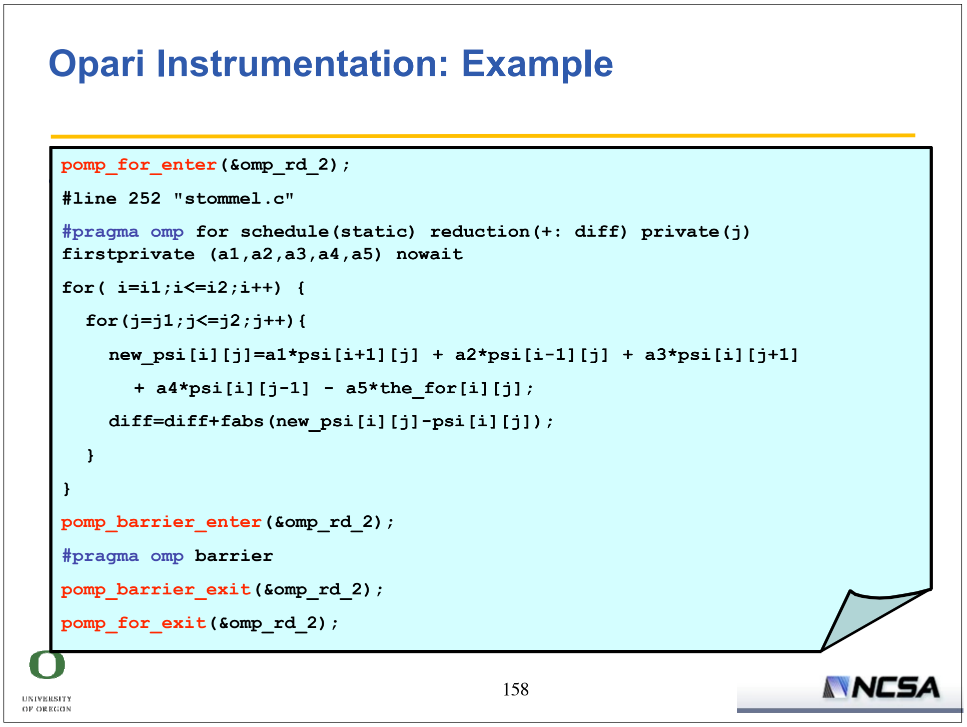
  `[# = init|destroy|set|unset|test]`

- POMP version
  - Calls omp version internally
  - Can do extra stuff before and after call



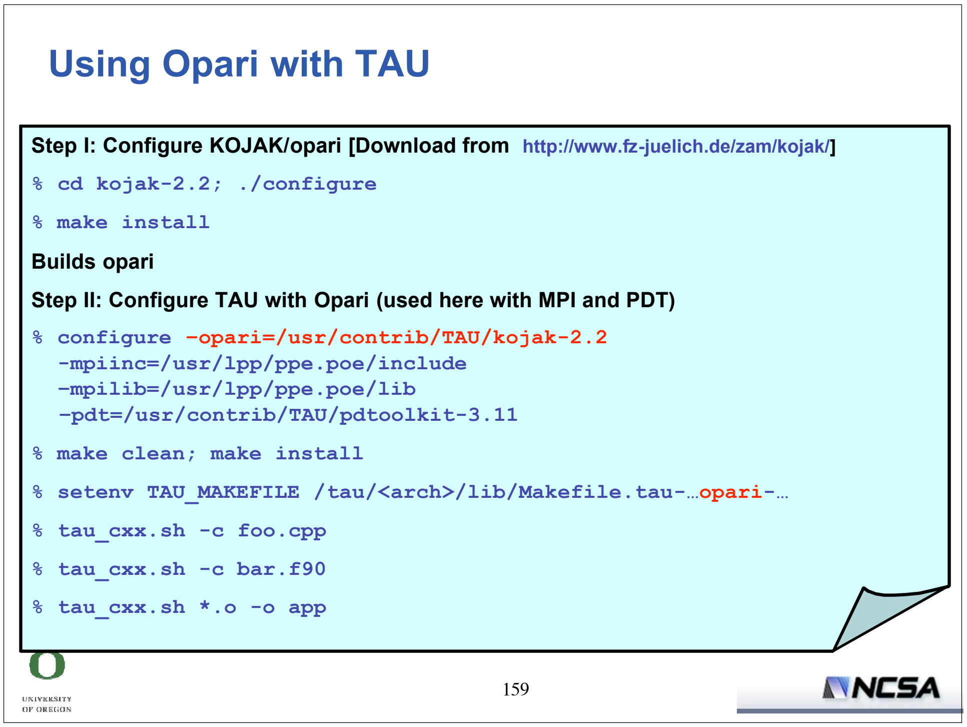
Example: !$OMP PARALLEL DO Instrumentation

```
call pomp_parallel_fork(d)
!$OMP PARALLEL other-clauses...
      call pomp_parallel_begin(d)
      call pomp_do_enter(d)
      !$OMP DO schedule-clauses, ordered-clauses,
               lastprivate-clauses
            do loop
      !$OMP END DO NOWAIT
      call pomp_barrier_enter(d)
      !$OMP BARRIER
      call pomp_barrier_exit(d)
      call pomp_do_exit(d)
      call pomp_parallel_end(d)
!$OMP END PARALLEL DO
call pomp_parallel_join(d)
```



Opari Instrumentation: Example

```
pomp_for_enter(&omp_rd_2);
#line 252 "stommel.c"
#pragma omp for schedule(static) reduction(+: diff) private(j)
firstprivate (a1,a2,a3,a4,a5) nowait
for( i=i1;i<=i2;i++) {
  for(j=j1;j<=j2;j++){
    new_psi[i][j]=a1*psi[i+1][j] + a2*psi[i-1][j] + a3*psi[i][j+1]
      + a4*psi[i][j-1] - a5*the_for[i][j];
    diff=diff+fabs(new_psi[i][j]-psi[i][j]);
  }
}
pomp_barrier_enter(&omp_rd_2);
#pragma omp barrier
pomp_barrier_exit(&omp_rd_2);
pomp_for_exit(&omp_rd_2);
```



Using Opari with TAU

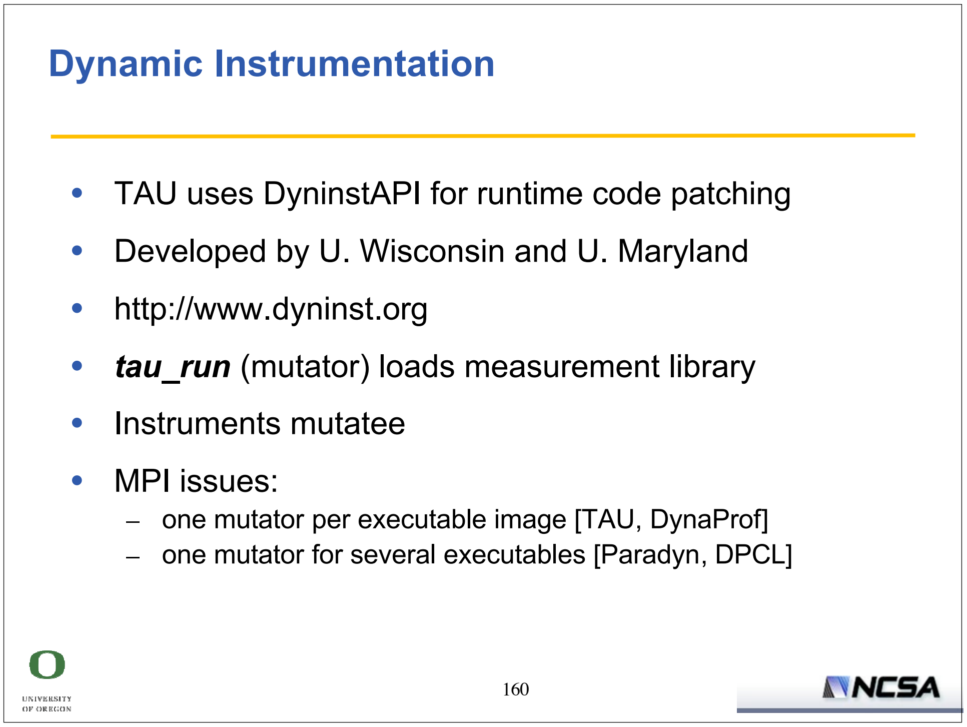
Step I: Configure KOJAK/opari [Download from http://www.fz-juelich.de/zam/kojak/]

```
% cd kojak-2.2; ./configure
% make install
```

Builds opari

Step II: Configure TAU with Opari (used here with MPI and PDT)

```
% configure -opari=/usr/contrib/TAU/kojak-2.2
  -mpiinc=/usr/lpp/ppe.poe/include
  -mpilib=/usr/lpp/ppe.poe/lib
  -pdt=/usr/contrib/TAU/pdtoolkit-3.11
% make clean; make install
% setenv TAU_MAKEFILE /tau/<arch>/lib/Makefile.tau-…opari-…
% tau_cxx.sh -c foo.cpp
% tau_cxx.sh -c bar.f90
% tau_cxx.sh *.o -o app
```



Dynamic Instrumentation

- TAU uses DyninstAPI for runtime code patching
- Developed by U. Wisconsin and U. Maryland
- http://www.dyninst.org
- ***tau_run*** (mutator) loads measurement library
- Instruments mutatee
- MPI issues:
  - one mutator per executable image [TAU, DynaProf]
  - one mutator for several executables [Paradyn, DPCL]

Using DyninstAPI with TAU

Step I: Install DyninstAPI[Download from http://www.dyninst.org]

```
% cd dyninstAPI-4.2.1/core; make
```

Set DyninstAPI environment variables (including LD_LIBRARY_PATH)

Step II: Configure TAU with Dyninst

```
% configure -dyninst=/usr/local/dyninstAPI-4.2.1
% make clean; make install
```

Builds <taudir>/<arch>/bin/tau_run

```
% tau_run [<-o outfile>] [-Xrun<libname>][-f <select_inst_file>] [-v] <infile>
% tau_run -o a.inst.out a.out
```

Rewrites a.out

```
% tau_run klargest
```

Instruments klargest with TAU calls and executes it

```
% tau_run -XrunTAUsh-papi a.out
```

Loads libTAUsh-papi.so instead of libTAU.so for measurements

Virtual Machine Performance Instrumentation

- Integrate performance system with VM
  - Captures robust performance data (e.g., thread events)
  - Maintain features of environment
    - portability, concurrency, extensibility, interoperation
  - Allow use in optimization methods
- JVM Profiling Interface (JVMPI)
  - Generation of JVM events and hooks into JVM
  - Profiler agent (TAU) loaded as shared object
    - registers events of interest and address of callback routine
  - Access to information on dynamically loaded classes
  - No need to modify Java source, bytecode, or JVM

Using TAU with Java Applications

Step I: Sun JDK 1.4+ [download from www.javasoft.com]

Step II: Configure TAU with JDK (v 1.2 or better)

```
% configure -jdk=/usr/java2 -TRACE -PROFILE
% make clean; make install
```

Builds <taudir>/<arch>/lib/libTAU.so

For Java (without instrumentation):

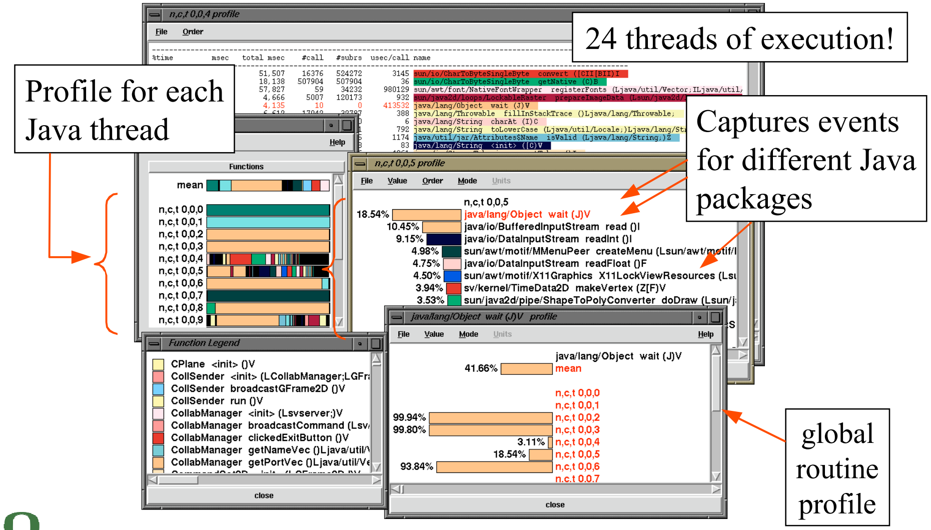
```
% java application
```

With instrumentation:

```
% java -XrunTAU application
% java -XrunTAU:exclude=sun/io,java application
```

Excludes sun/io/* and java/* classes

TAU Profiling of Java Application (SciVis)



24 threads of execution!

Profile for each Java thread

Captures events for different Java packages

global routine profile

n,c,t 0,0,4 profile

```
---------------------------------------------------------------------------
%time         msec   total msec    #call    #subrs  usec/call name
---------------------------------------------------------------------------
                        51,507     16376    524272       3145 sun/io/CharToByteSingleByte  convert ([CII[BII)I
                        18,138    507904    507904         36 sun/io/CharToByteSingleByte  getNative (C)B
                        57,827        59     34232     980129 sun/awt/font/NativeFontWrapper  registerFonts (Ljava/util/Vector;ILjava/util,
                         4,666      5007    120173        932 sun/java2d/loops/LockableRaster  prepareImageData (Lsun/java2d/
                         4,135        10         0     413532 java/lang/Object  wait (J)V
                                                          388 java/lang/Throwable  fillInStackTrace ()Ljava/lang/Throwable;
                                                            6 java/lang/String  charAt (I)C
                                                          792 java/lang/String  toLowerCase (Ljava/util/Locale;)Ljava/lang/Str
                                                         1174 java/util/jar/Attributes$Name  isValid (Ljava/lang/String;)Z
                                                           83 java/lang/String  <init> ([C)V
```

Function Legend

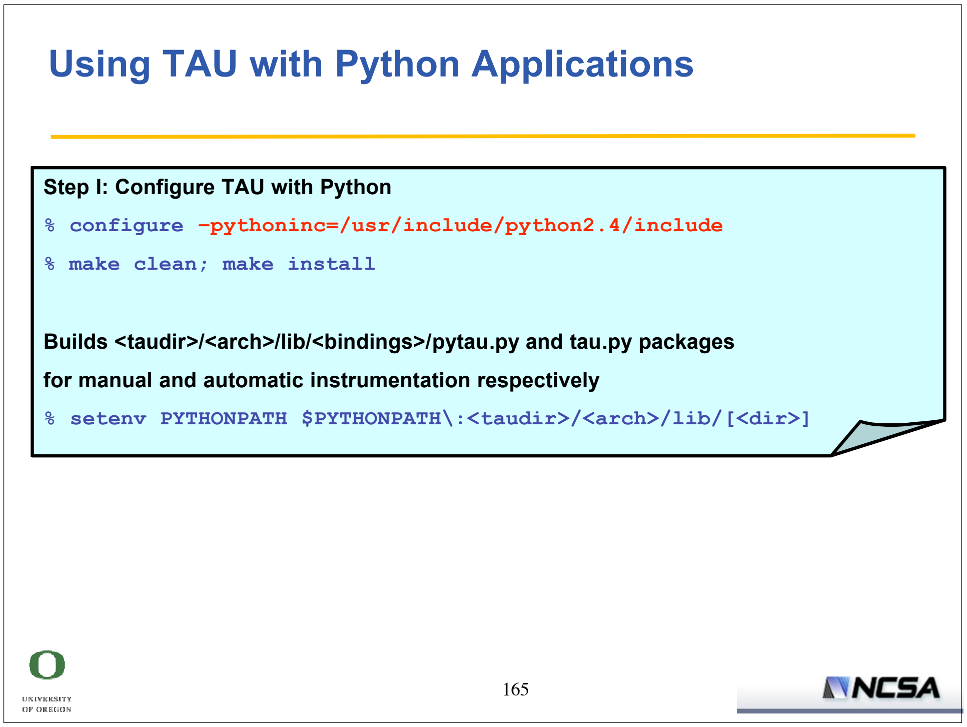
- CPlane <init> ()V
- CollSender <init> (LCollabManager;LGFra
- CollSender broadcastGFrame2D ()V
- CollSender run ()V
- CollabManager <init> (Lsvserver;)V
- CollabManager broadcastCommand (Lsv
- CollabManager clickedExitButton ()V
- CollabManager getNameVec ()Ljava/util/V
- CollabManager getPortVec ()Ljava/util/Ve

n,c,t 0,0,5 profile

	n,c,t 0,0,5
18.54%	java/lang/Object wait (J)V
10.45%	java/io/BufferedInputStream read ()I
9.15%	java/io/DataInputStream readInt ()I
4.98%	sun/awt/motif/MMenuPeer createMenu (Lsun/awt/motif/I
4.75%	java/io/DataInputStream readFloat ()F
4.50%	sun/awt/motif/X11Graphics X11LockViewResources (Lsu
3.94%	sv/kernel/TimeData2D makeVertex (Z[F)V
3.53%	sun/java2d/pipe/ShapeToPolyConverter doDraw (Lsun/j

java/lang/Object wait (J)V profile

	java/lang/Object wait (J)V
41.66%	mean
	n,c,t 0,0,0
	n,c,t 0,0,1
99.94%	n,c,t 0,0,2
99.80%	n,c,t 0,0,3
3.11%	n,c,t 0,0,4
18.54%	n,c,t 0,0,5
93.84%	n,c,t 0,0,6
	n,c,t 0,0,7



Using TAU with Python Applications

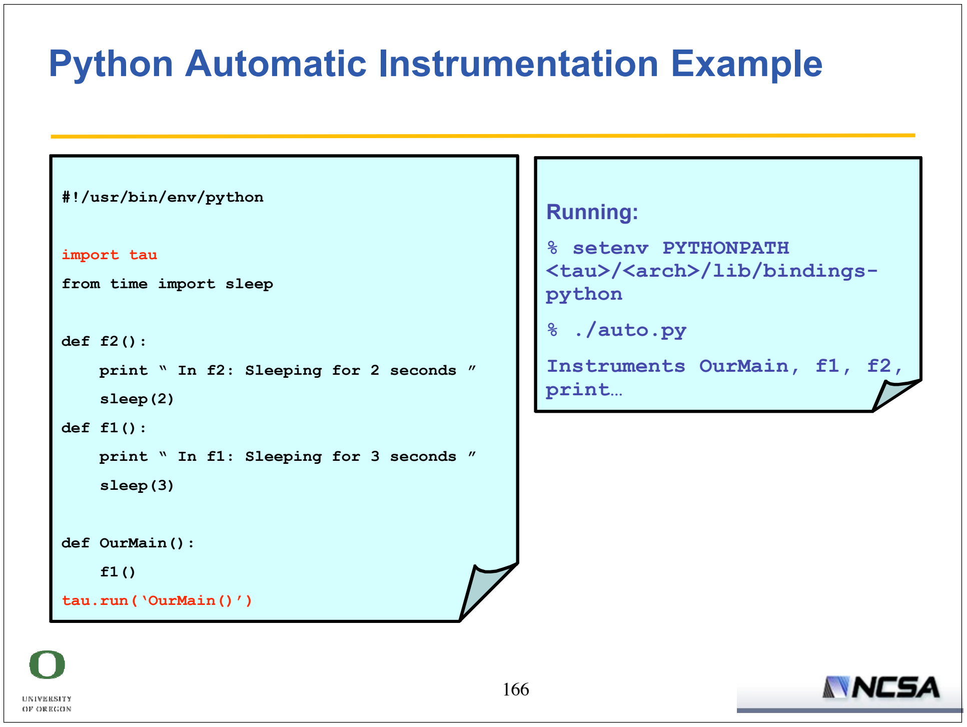
Step I: Configure TAU with Python

```
% configure -pythoninc=/usr/include/python2.4/include
% make clean; make install
```

Builds <taudir>/<arch>/lib/<bindings>/pytau.py and tau.py packages

for manual and automatic instrumentation respectively

```
% setenv PYTHONPATH $PYTHONPATH\:<taudir>/<arch>/lib/[<dir>]
```



Python Automatic Instrumentation Example

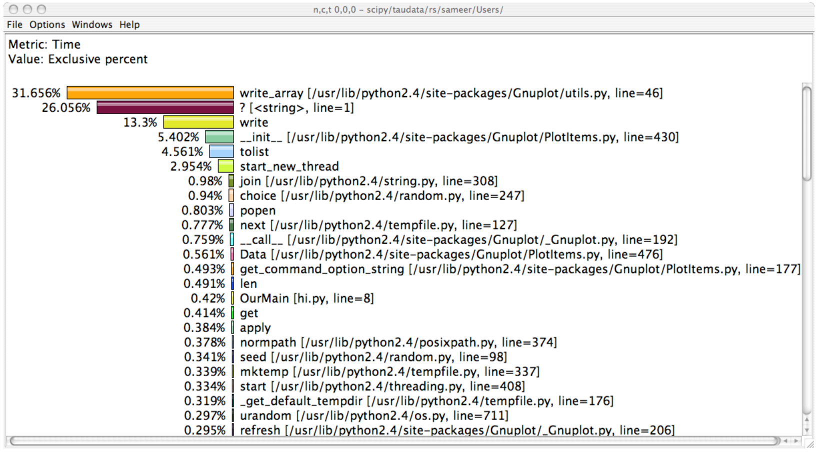
```
#!/usr/bin/env/python

import tau
from time import sleep

def f2():
    print “ In f2: Sleeping for 2 seconds ”
    sleep(2)
def f1():
    print “ In f1: Sleeping for 3 seconds ”
    sleep(3)

def OurMain():
    f1()
tau.run(‘OurMain()’)
```

Running:

```
% setenv PYTHONPATH
<tau>/<arch>/lib/bindings-
python
% ./auto.py
```

Instruments OurMain, f1, f2, print…

Python Instrumentation: SciPy

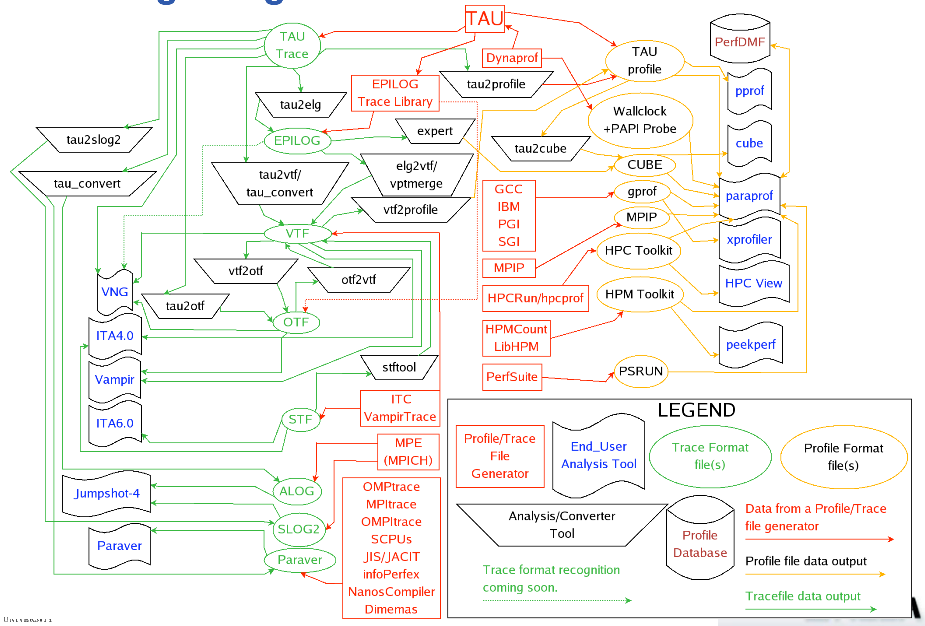

n,c,t 0,0,0 – scipy/taudata/rs/sameer/Users/
File Options Windows Help

```
Metric: Time
Value: Exclusive percent

31.656%  write_array [/usr/lib/python2.4/site-packages/Gnuplot/utils.py, line=46]
26.056%  ? [<string>, line=1]
13.3%    write
5.402%   __init__ [/usr/lib/python2.4/site-packages/Gnuplot/PlotItems.py, line=430]
4.561%   tolist
2.954%   start_new_thread
0.98%    join [/usr/lib/python2.4/string.py, line=308]
0.94%    choice [/usr/lib/python2.4/random.py, line=247]
0.803%   popen
0.777%   next [/usr/lib/python2.4/tempfile.py, line=127]
0.759%   __call__ [/usr/lib/python2.4/site-packages/Gnuplot/_Gnuplot.py, line=192]
0.561%   Data [/usr/lib/python2.4/site-packages/Gnuplot/PlotItems.py, line=476]
0.493%   get_command_option_string [/usr/lib/python2.4/site-packages/Gnuplot/PlotItems.py, line=177]
0.491%   len
0.42%    OurMain [hi.py, line=8]
0.414%   get
0.384%   apply
0.378%   normpath [/usr/lib/python2.4/posixpath.py, line=374]
0.341%   seed [/usr/lib/python2.4/random.py, line=98]
0.339%   mktemp [/usr/lib/python2.4/tempfile.py, line=337]
0.334%   start [/usr/lib/python2.4/threading.py, line=408]
0.319%   _get_default_tempdir [/usr/lib/python2.4/tempfile.py, line=176]
0.297%   urandom [/usr/lib/python2.4/os.py, line=711]
0.295%   refresh [/usr/lib/python2.4/site-packages/Gnuplot/_Gnuplot.py, line=206]
```

Performance Analysis

- paraprof profile browser (GUI)
- pprof (text based profile browser)
- TAU traces can be exported to many different tools
  - Vampir/VNG [T.U. Dresden] (formerly Intel (R) Trace Analyzer)
  - EXPERT [FZJ]
  - Jumpshot (bundled with TAU) [Argonne National Lab] ...

Building Bridges to Other Tools: TAU



TAU
Dynaprof
TAU Trace
TAU profile
PerfDMF
EPILOG Trace Library
tau2profile
pprof
tau2elg
Wallclock +PAPI Probe
tau2slog2
expert
EPILOG
tau2cube
cube
tau2vtf/ tau_convert
elg2vtf/ vptmerge
CUBE
tau_convert
GCC IBM PGI SGI
gprof
paraprof
vtf2profile
VTF
MPIP
xprofiler
MPIP
HPC Toolkit
vtf2otf
otf2vtf
VNG
HPCRun/hpcprof
HPM Toolkit
HPC View
tau2otf
OTF
ITA4.0
HPMCount LibHPM
peekperf
stftool
Vampir
PerfSuite
PSRUN
ITC VampirTrace
ITA6.0
STF
MPE (MPICH)
Jumpshot-4
ALOG
OMPtrace MPItrace OMPItrace SCPUs JIS/JACIT infoPerfex NanosCompiler Dimemas
SLOG2
Paraver
Paraver

LEGEND
Profile/Trace File Generator
End_User Analysis Tool
Trace Format file(s)
Profile Format file(s)
Analysis/Converter Tool
Profile Database
Data from a Profile/Trace file generator
Profile file data output
Trace format recognition coming soon.
Tracefile data output

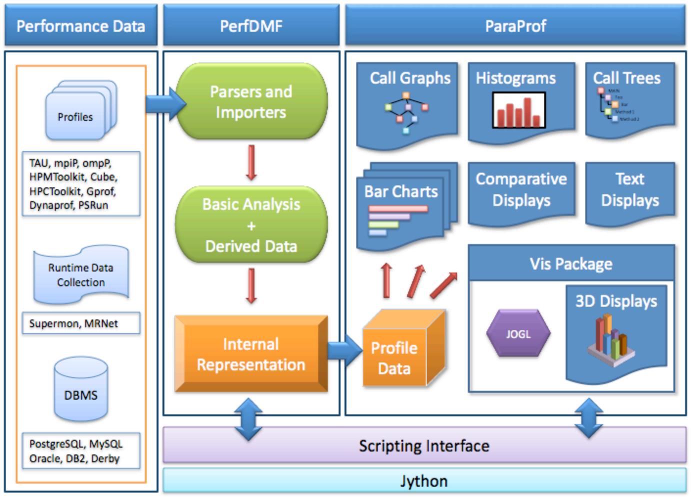
TAU Performance System Interfaces

- PDT [U. Oregon, LANL, FZJ] for instrumentation of C++, C99, F95 source code
- PAPI [UTK] for accessing hardware performance counters data
- DyninstAPI [U. Maryland, U. Wisconsin] for runtime instrumentation
- KOJAK [FZJ, UTK]
  - Epilog trace generation library
  - CUBE callgraph visualizer
  - Opari OpenMP directive rewriting tool
- Vampir/VNG Trace Analyzer [TU Dresden]
- VTF3/OTF trace generation library [TU Dresden] (available from TAU website)
- Paraver trace visualizer [CEPBA]
- Jumpshot-4 trace visualizer [MPICH, ANL]
- JVMPI from JDK for Java program instrumentation [Sun]
- Paraprof profile browser/PerfDMF database supports:
  - TAU format
  - Gprof [GNU]
  - HPM Toolkit [IBM]
  - MpiP [ORNL, LLNL]
  - Dynaprof [UTK]
  - PSRun [NCSA]

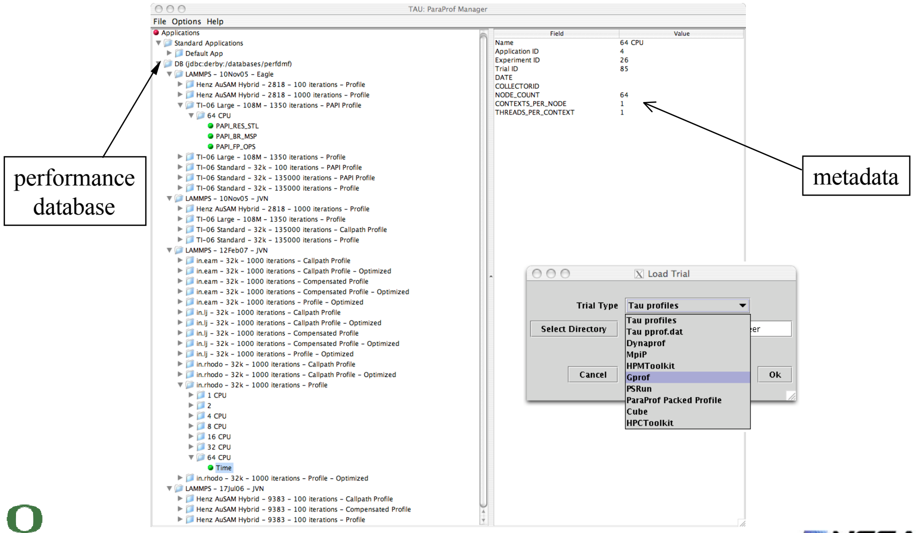
Performance Analysis and Visualization

- Analysis of parallel profile and trace measurement
- **Parallel profile analysis (*ParaProf*)**
  - Java-based analysis and visualization tool
  - Support for large-scale parallel profiles
- Performance data management framework (***PerfDMF***)
- **Parallel trace analysis**
  - Translation to *VTF* (V3.0), *EPILOG, OTF* formats
  - Integration with *Vampir / Vampir Server* (TU Dresden)
  - Profile generation from trace data
- Online parallel analysis and visualization
- Integration with *CUBE* browser (KOJAK, UTK, FZJ)

ParaProf – Parallel Performance Profile Analysis



ParaProf – Manager Window



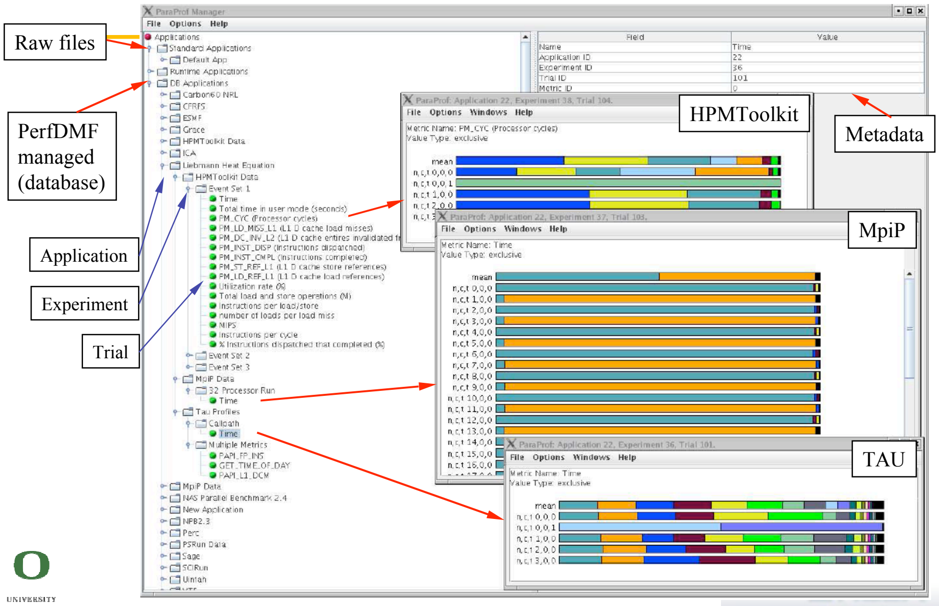
```
Applications
  Standard Applications
    Default App
  DB (jdbc:derby:/databases/perfdmf)
    LAMMPS – 10Nov05 – Eagle
      Henz AuSAM Hybrid – 2818 – 100 iterations – Profile
      Henz AuSAM Hybrid – 2818 – 1000 iterations – Profile
      TI-06 Large – 108M – 1350 iterations – PAPI Profile
        64 CPU
          PAPI_RES_STL
          PAPI_BR_MSP
          PAPI_FP_OPS
      TI-06 Large – 108M – 1350 iterations – Profile
      TI-06 Standard – 32k – 100 iterations – PAPI Profile
      TI-06 Standard – 32k – 135000 iterations – PAPI Profile
      TI-06 Standard – 32k – 135000 iterations – Profile
    LAMMPS – 10Nov05 – JVN
      Henz AuSAM Hybrid – 2818 – 1000 iterations – Profile
      TI-06 Large – 108M – 1350 iterations – Profile
      TI-06 Standard – 32k – 135000 iterations – Callpath Profile
      TI-06 Standard – 32k – 135000 iterations – Profile
    LAMMPS – 12Feb07 – JVN
      in.eam – 32k – 1000 iterations – Callpath Profile
      in.eam – 32k – 1000 iterations – Callpath Profile – Optimized
      in.eam – 32k – 1000 iterations – Compensated Profile
      in.eam – 32k – 1000 iterations – Compensated Profile – Optimized
      in.eam – 32k – 1000 iterations – Profile – Optimized
      in.lj – 32k – 1000 iterations – Callpath Profile
      in.lj – 32k – 1000 iterations – Callpath Profile – Optimized
      in.lj – 32k – 1000 iterations – Compensated Profile
      in.lj – 32k – 1000 iterations – Compensated Profile – Optimized
      in.lj – 32k – 1000 iterations – Profile – Optimized
      in.rhodo – 32k – 1000 iterations – Callpath Profile
      in.rhodo – 32k – 1000 iterations – Callpath Profile – Optimized
      in.rhodo – 32k – 1000 iterations – Profile
        1 CPU
        2
        4 CPU
        8 CPU
        16 CPU
        32 CPU
        64 CPU
          Time
      in.rhodo – 32k – 1000 iterations – Profile – Optimized
    LAMMPS – 17Jul06 – JVN
      Henz AuSAM Hybrid – 9383 – 100 iterations – Callpath Profile
      Henz AuSAM Hybrid – 9383 – 100 iterations – Compensated Profile
      Henz AuSAM Hybrid – 9383 – 100 iterations – Profile
```

Field	Value
Name	64 CPU
Application ID	4
Experiment ID	26
Trial ID	85
DATE	
COLLECTORID	
NODE_COUNT	64
CONTEXTS_PER_NODE	1
THREADS_PER_CONTEXT	1

Trial Type: Tau profiles

- Tau profiles
- Tau pprof.dat
- Dynaprof
- MpiP
- HPMToolkit
- Gprof
- PSRun
- ParaProf Packed Profile
- Cube
- HPCToolkit

ParaProf – Manager Window



```
Applications
  Standard Applications
    Default App
  Runtime Applications
  DB Applications
    Carbon60 NRL
    CFRFS
    ESMF
    Grace
    HPMToolkit Data
    ICA
    Liebmann Heat Equation
      HPMToolkit Data
        Event Set 1
          Time
          Total time in user mode (seconds)
          PM_CYC (Processor cycles)
          PM_LD_MISS_L1 (L1 D cache load misses)
          PM_DC_INV_L2 (L1 D cache entires invalidated fr
          PM_INST_DISP (instructions dispatched)
          PM_INST_CMPL (instructions completed)
          PM_ST_REF_L1 (L1 D cache store references)
          PM_LD_REF_L1 (L1 D cache load references)
          Utilization rate (%)
          Total load and store operations (M)
          Instructions per load/store
          number of loads per load miss
          MIPS
          Instructions per cycle
          % Instructions dispatched that completed (%)
        Event Set 2
        Event Set 3
      MpiP Data
        32 Processor Run
          Time
      Tau Profiles
        Callpath
          Time
        Multiple Metrics
          PAPI_FP_INS
          GET_TIME_OF_DAY
          PAPI_L1_DCM
    MpiP Data
    NAS Parallel Benchmark 2.4
    New Application
    NPB2.3
    Perc
    PSRun Data
    Sage
    SCIRun
    Uintah
```

Field	Value
Name	Time
Application ID	22
Experiment ID	36
Trial ID	101
Metric ID	0

Metric Name: PM_CYC (Processor cycles)
Value Type: exclusive

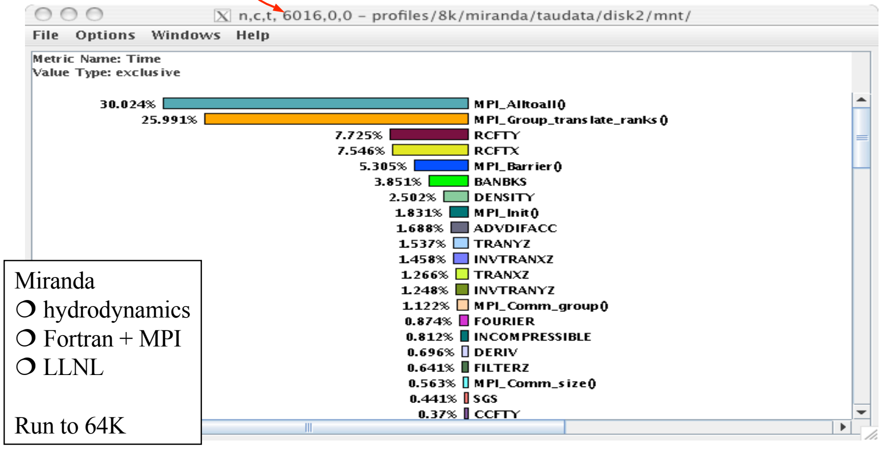
Metric Name: Time
Value Type: exclusive

Metric Name: Time
Value Type: exclusive

ParaProf – Flat Profile (Miranda, BG/L)

node, context, thread

8K processors

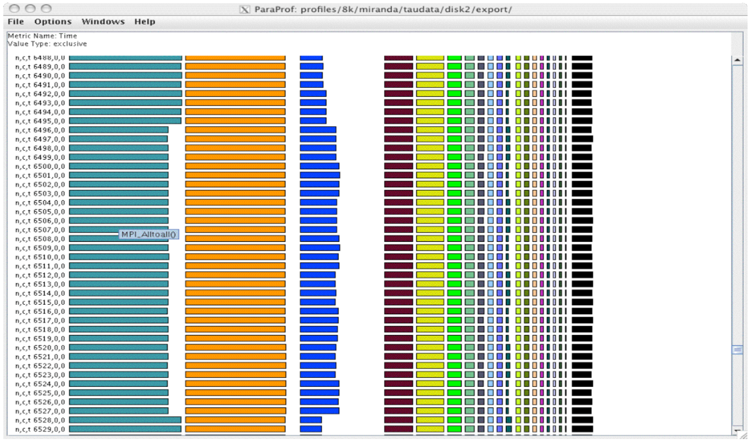


Metric Name: Time
Value Type: exclusive

Miranda
- hydrodynamics
- Fortran + MPI
- LLNL

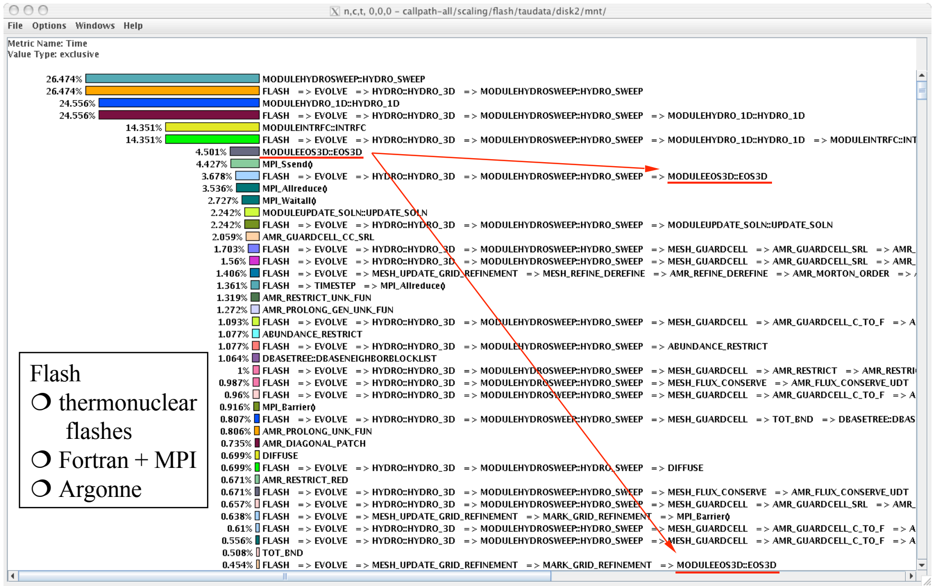
Run to 64K

ParaProf – Stacked View (Miranda)



Metric Name: Time
Value Type: exclusive

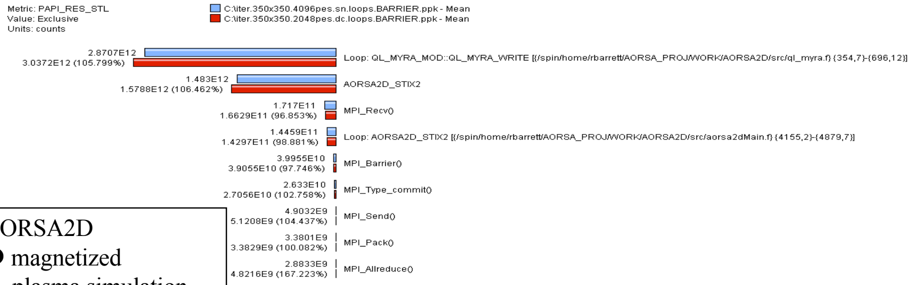
ParaProf – Callpath Profile (Flash)



```
n,c,t, 0,0,0 - callpath-all/scaling/flash/taudata/disk2/mnt/
File  Options  Windows  Help
Metric Name: Time
Value Type: exclusive

 26.474% MODULEHYDROSWEEP::HYDRO_SWEEP
 26.474% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP
 24.556% MODULEHYDRO_1D::HYDRO_1D
 24.556% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEHYDRO_1D::HYDRO_1D
 14.351% MODULEINTRFC::INTRFC
 14.351% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEHYDRO_1D::HYDRO_1D  => MODULEINTRFC::INT
  4.501% MODULEEOS3D::EOS3D
  4.427% MPI_Ssend()
  3.678% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEEOS3D::EOS3D
  3.536% MPI_Allreduce()
  2.727% MPI_Waitall()
  2.242% MODULEUPDATE_SOLN::UPDATE_SOLN
  2.242% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MODULEUPDATE_SOLN::UPDATE_SOLN
  2.059% AMR_GUARDCELL_CC_SRL
  1.703% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_SRL  => AMR_
   1.56% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_SRL  => AMR_
  1.406% FLASH  => EVOLVE  => MESH_UPDATE_GRID_REFINEMENT  => MESH_REFINE_DEREFINE  => AMR_REFINE_DEREFINE  => AMR_MORTON_ORDER  =>
  1.361% FLASH  => TIMESTEP  => MPI_Allreduce()
  1.319% AMR_RESTRICT_UNK_FUN
  1.272% AMR_PROLONG_GEN_UNK_FUN
  1.093% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
  1.077% ABUNDANCE_RESTRICT
  1.077% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => ABUNDANCE_RESTRICT
  1.064% DBASETREE::DBASENEIGHBORBLOCKLIST
     1% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_RESTRICT  => AMR_RESTRI
  0.987% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_FLUX_CONSERVE  => AMR_FLUX_CONSERVE_UDT
   0.96% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
  0.916% MPI_Barrier()
  0.807% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => TOT_BND  => DBASETREE::DBAS
  0.806% AMR_PROLONG_UNK_FUN
  0.735% AMR_DIAGONAL_PATCH
  0.699% DIFFUSE
  0.699% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => DIFFUSE
  0.671% AMR_RESTRICT_RED
  0.671% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_FLUX_CONSERVE  => AMR_FLUX_CONSERVE_UDT
  0.657% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_SRL  => AMR_
  0.638% FLASH  => EVOLVE  => MESH_UPDATE_GRID_REFINEMENT  => MARK_GRID_REFINEMENT  => MPI_Barrier()
   0.61% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
  0.556% FLASH  => EVOLVE  => HYDRO::HYDRO_3D  => MODULEHYDROSWEEP::HYDRO_SWEEP  => MESH_GUARDCELL  => AMR_GUARDCELL_C_TO_F  => A
  0.508% TOT_BND
  0.454% FLASH  => EVOLVE  => MESH_UPDATE_GRID_REFINEMENT  => MARK_GRID_REFINEMENT  => MODULEEOS3D::EOS3D
```

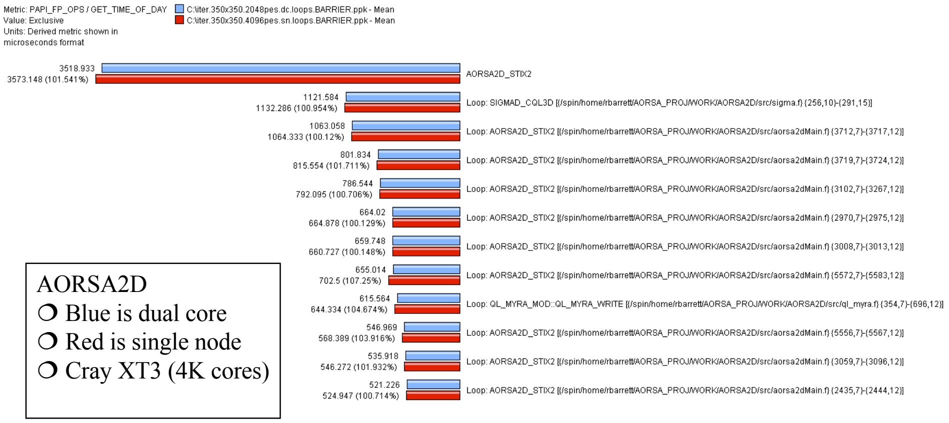
Flash
- ❍ thermonuclear flashes
- ❍ Fortran + MPI
- ❍ Argonne

Comparing Effects of Multi-Core Processors



AORSA2D
- ❍ magnetized plasma simulation
- ❍ Blue is single node
- ❍ Red is dual core
- ❍ Cray XT3 (4K cores)

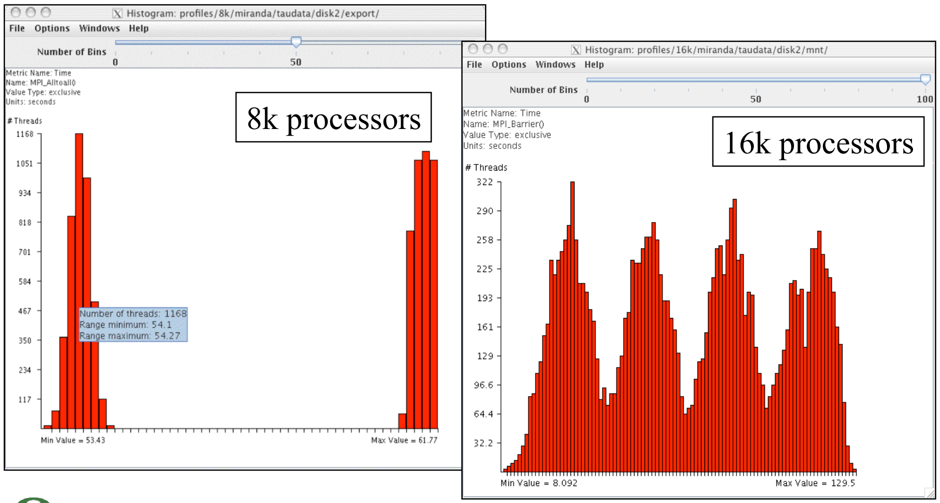
Comparing FLOPS (AORSA2D, Cray XT3)


Metric: PAPI_FP_OPS / GET_TIME_OF_DAY
Value: Exclusive
Units: Derived metric shown in microseconds format

AORSA2D

- Blue is dual core
- Red is single node
- Cray XT3 (4K cores)

ParaProf – Scalable Histogram View (Miranda)



Metric Name: Time
Name: MPI_Alltoall()
Value Type: exclusive
Units: seconds

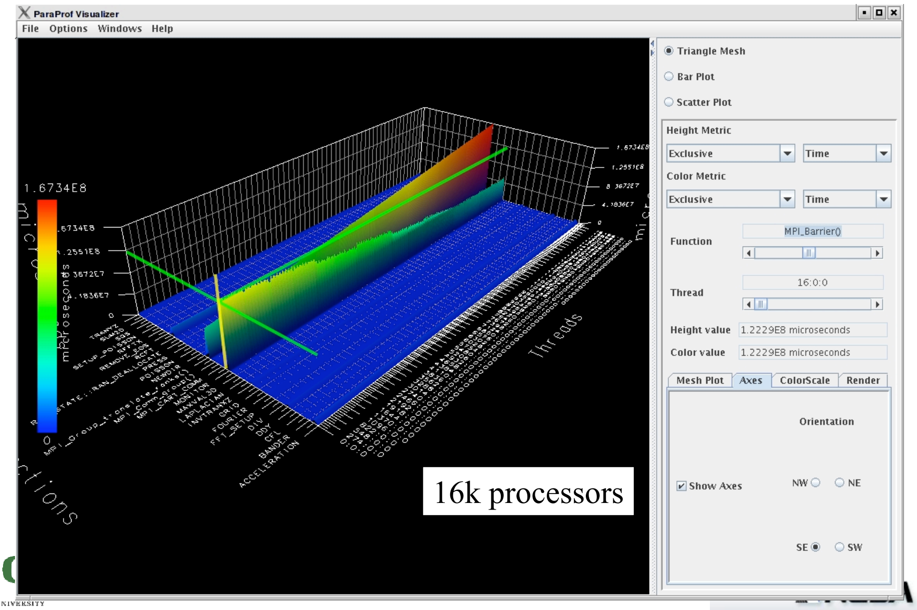
Number of threads: 1168
Range minimum: 54.1
Range maximum: 54.27

8k processors

Metric Name: Time
Name: MPI_Barrier()
Value Type: exclusive
Units: seconds

16k processors

ParaProf – Full Profile (Miranda)



● Triangle Mesh
○ Bar Plot
○ Scatter Plot

Height Metric: Exclusive | Time

Color Metric: Exclusive | Time

Function: MPI_Barrier()

Thread: 16:0:0

Height value: 1.2229E8 microseconds

Color value: 1.2229E8 microseconds

Mesh Plot | Axes | ColorScale | Render

Orientation

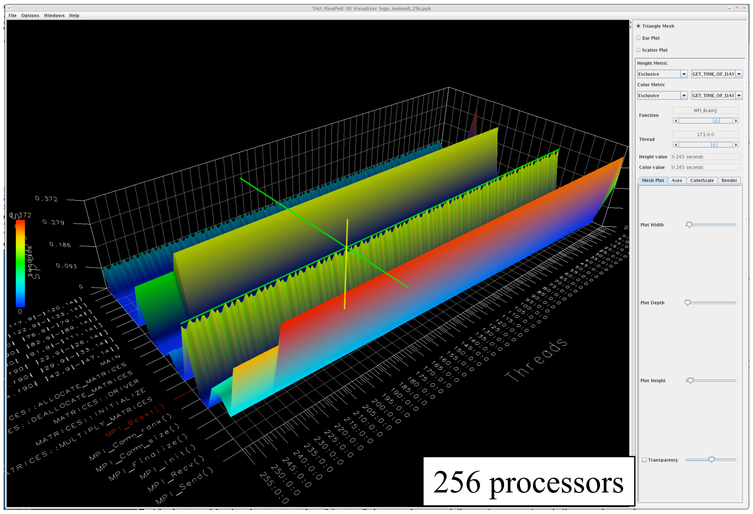
☑ Show Axes

NW ○ ○ NE

SE ● ○ SW

16k processors

ParaProf –Full Profile (Matmult, ANL BGP)



● Triangle Mesh
○ Bar Plot
○ Scatter Plot

Height Metric: Exclusive | GET_TIME_OF_DAY

Color Metric: Exclusive | GET_TIME_OF_DAY

Function: MPI_Bcast()

Thread: 171:0:0

Height value: 0.263 seconds

Color value: 0.263 seconds

Mesh Plot | Axes | ColorScale | Render

Plot Width

Plot Depth

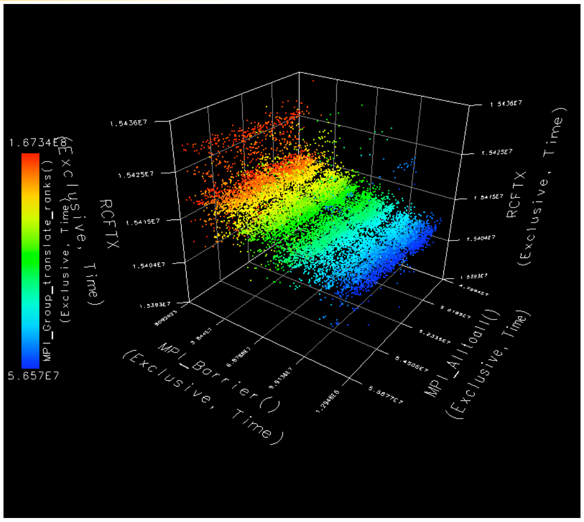
Plot Height

□ Transparency

256 processors

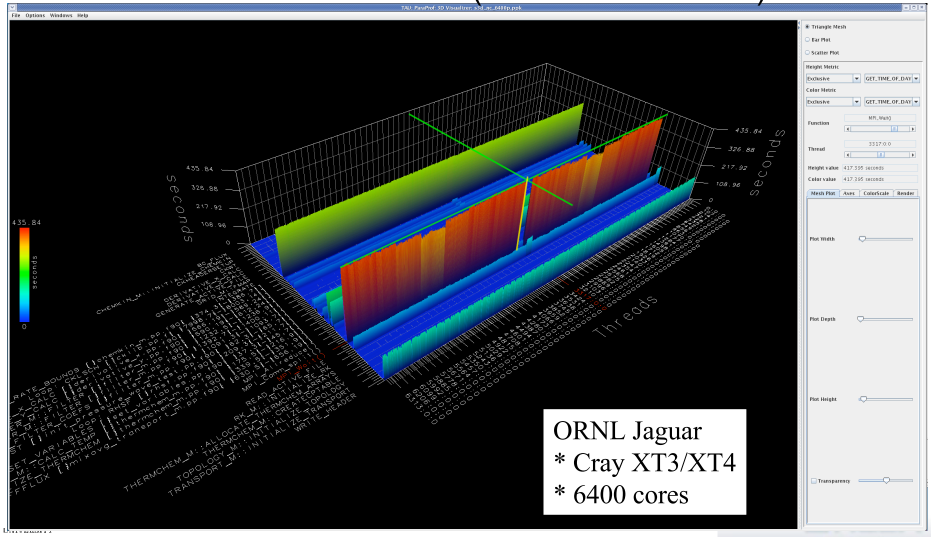
ParaProf – 3D Scatterplot (Miranda)

- Each point is a "thread" of execution
- A total of four metrics shown in relation
- ParaProf's visualization library
  - JOGL



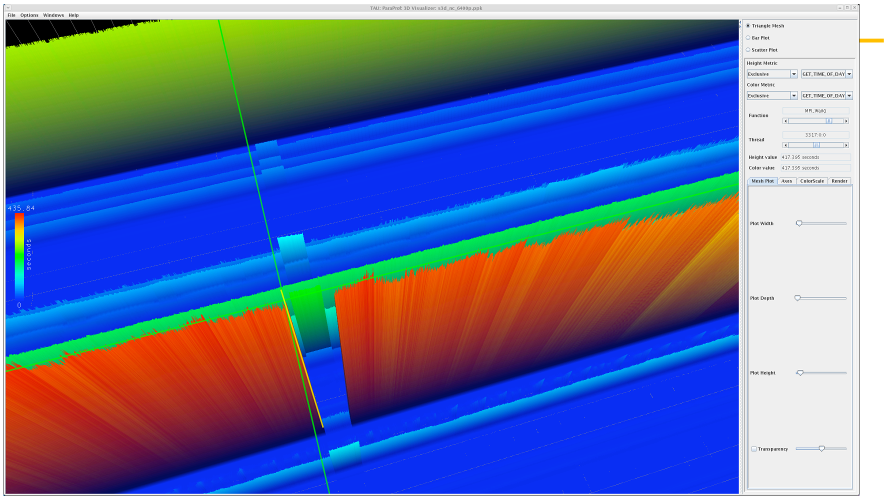
Visualizing Hybrid Problems (S3D, XT3+XT4)

- S3D combustion simulation (DOE SciDAC PERI)



ORNL Jaguar
* Cray XT3/XT4
* 6400 cores

Zoom View of Hybrid Execution (S3D, XT3+XT4)



● Triangle Mesh
○ Bar Plot
○ Scatter Plot

Height Metric: Exclusive, GET_TIME_OF_DAY

Color Metric: Exclusive, GET_TIME_OF_DAY

Function: MPI_Wait()

Thread: 3317:0:0

Height value: 417.395 seconds

Color value: 417.395 seconds

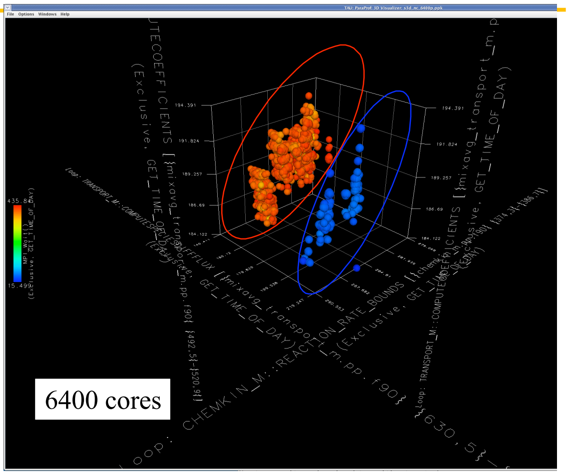
- Gap represents XT3 nodes
  - MPI_Wait takes less time, other routines take more time

UNIVERSITY OF OREGON

NCSA

Visualizing Hybrid Execution (S3D, XT3+XT4)

- Hybrid execution
- Process metadata is used to map performance to machine type
- Memory speed accounts for performance difference



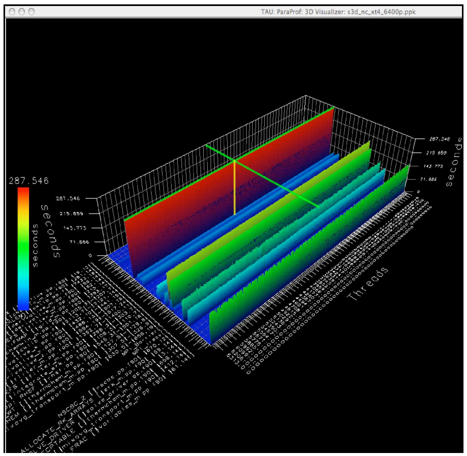
6400 cores

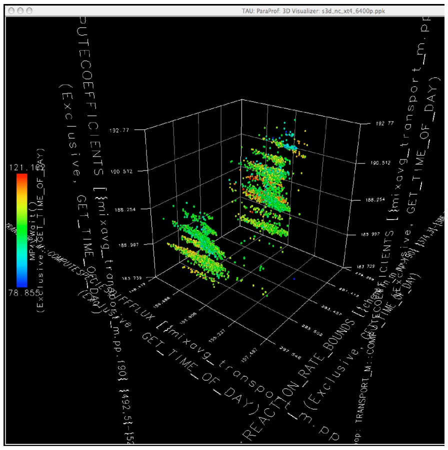
UNIVERSITY OF OREGON

NCSA

S3D Run on XT4 Only

- Better balance across nodes
- More performance uniformity





UNIVERSITY OF OREGON

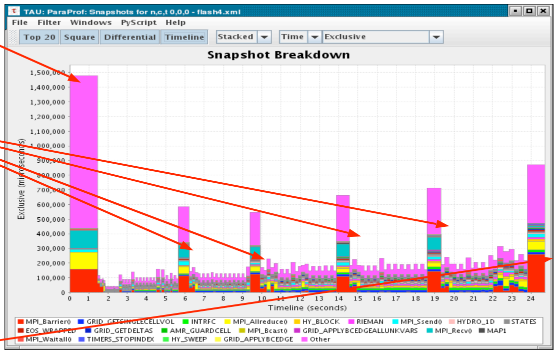
NCSA

ParaProf – Profile Snapshots (Flash)

- Profile snapshots are parallel profiles recorded at runtime
- Used to highlight profile changes during execution

Initialization

Checkpointing



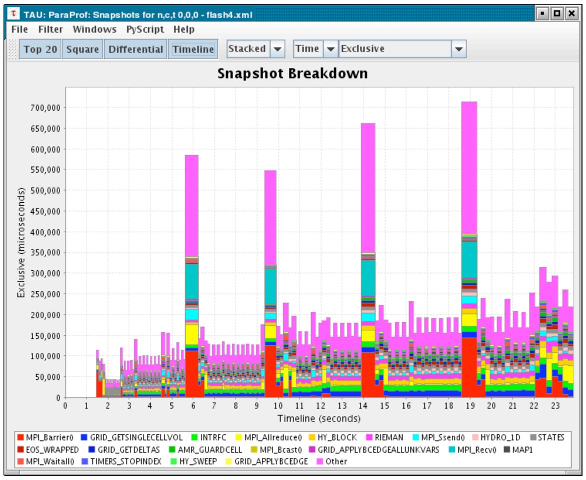
Finalization

UNIVERSITY OF OREGON

NCSA

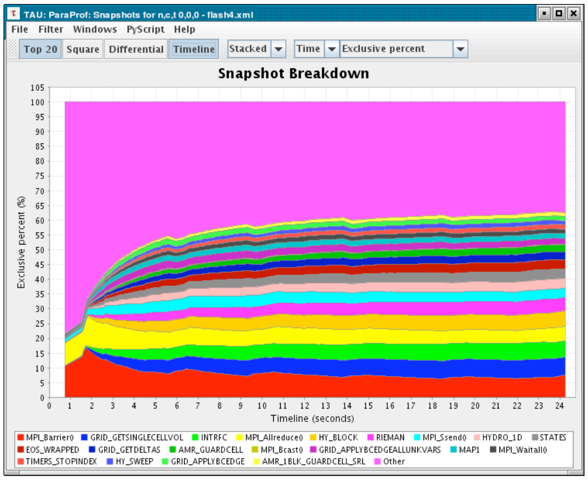
Filtered Profile Snapshots (Flash)

- Only show main loop iterations

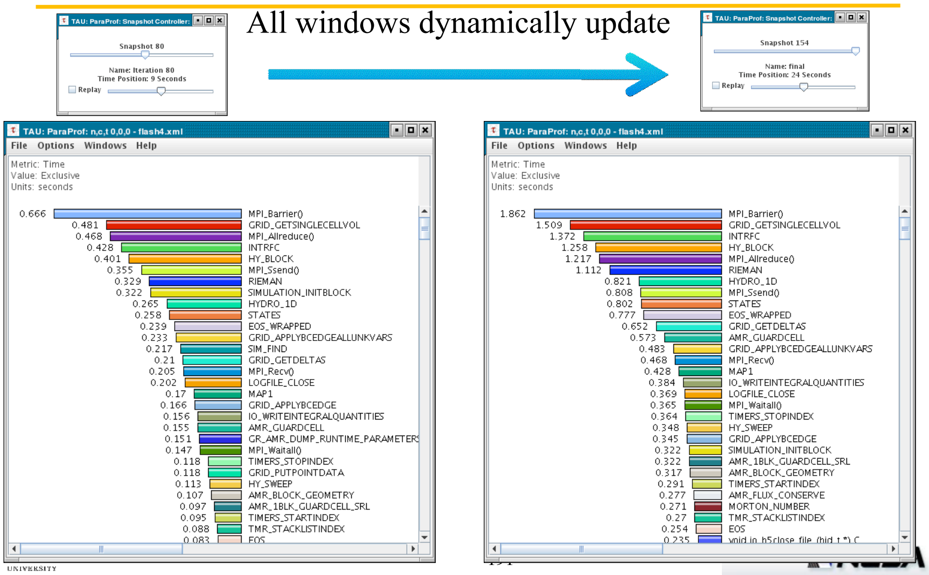


Profile Snapshots with Breakdown (Flash)

- Breakdown as a percentage



Profile Snapshot Replay (Flash)



All windows dynamically update

TAU: ParaProf: Snapshot Controller:

Snapshot 80

Name: Iteration 80
Time Position: 9 Seconds

□ Replay

TAU: ParaProf: Snapshot Controller:

Snapshot 154

Name: final
Time Position: 24 Seconds

□ Replay

TAU: ParaProf: n,c,t 0,0,0 - flash4.xml

Metric: Time
Value: Exclusive
Units: seconds

0.666	MPI_Barrier()
0.481	GRID_GETSINGLECELLVOL
0.468	MPI_Allreduce()
0.428	INTRFC
0.401	HY_BLOCK
0.355	MPI_Ssend()
0.329	RIEMAN
0.322	SIMULATION_INITBLOCK
0.265	HYDRO_1D
0.258	STATES
0.239	EOS_WRAPPED
0.233	GRID_APPLYBCEDGEALLUNKVARS
0.217	SIM_FIND
0.21	GRID_GETDELTAS
0.205	MPI_Recv()
0.202	LOGFILE_CLOSE
0.17	MAP1
0.166	GRID_APPLYBCEDGE
0.156	IO_WRITEINTEGRALQUANTITIES
0.155	AMR_GUARDCELL
0.151	GR_AMR_DUMP_RUNTIME_PARAMETERS
0.147	MPI_Waitall()
0.118	TIMERS_STOPINDEX
0.118	GRID_PUTPOINTDATA
0.113	HY_SWEEP
0.107	AMR_BLOCK_GEOMETRY
0.097	AMR_1BLK_GUARDCELL_SRL
0.095	TIMERS_STARTINDEX
0.088	TMR_STACKLISTINDEX
0.083	EOS

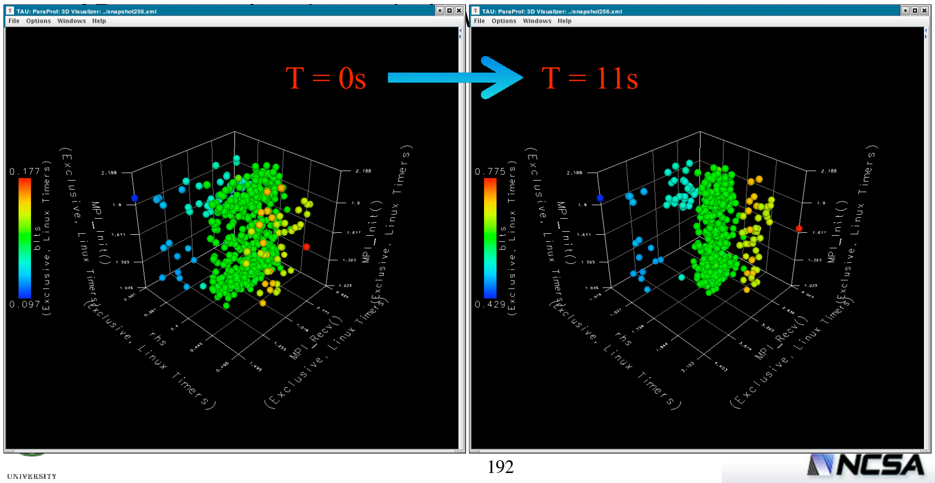
TAU: ParaProf: n,c,t 0,0,0 - flash4.xml

Metric: Time
Value: Exclusive
Units: seconds

1.862	MPI_Barrier()
1.509	GRID_GETSINGLECELLVOL
1.372	INTRFC
1.258	HY_BLOCK
1.217	MPI_Allreduce()
1.112	RIEMAN
0.821	HYDRO_1D
0.808	MPI_Ssend()
0.802	STATES
0.777	EOS_WRAPPED
0.652	GRID_GETDELTAS
0.573	AMR_GUARDCELL
0.483	GRID_APPLYBCEDGEALLUNKVARS
0.468	MPI_Recv()
0.428	MAP1
0.384	IO_WRITEINTEGRALQUANTITIES
0.369	LOGFILE_CLOSE
0.365	MPI_Waitall()
0.364	TIMERS_STOPINDEX
0.348	HY_SWEEP
0.345	GRID_APPLYBCEDGE
0.322	SIMULATION_INITBLOCK
0.322	AMR_1BLK_GUARDCELL_SRL
0.317	AMR_BLOCK_GEOMETRY
0.291	TIMERS_STARTINDEX
0.277	AMR_FLUX_CONSERVE
0.271	MORTON_NUMBER
0.27	TMR_STACKLISTINDEX
0.254	EOS
0.235	void io_h5close_file_(hid_t *) C

Snapshot Dynamics of Event Relations (Flash)

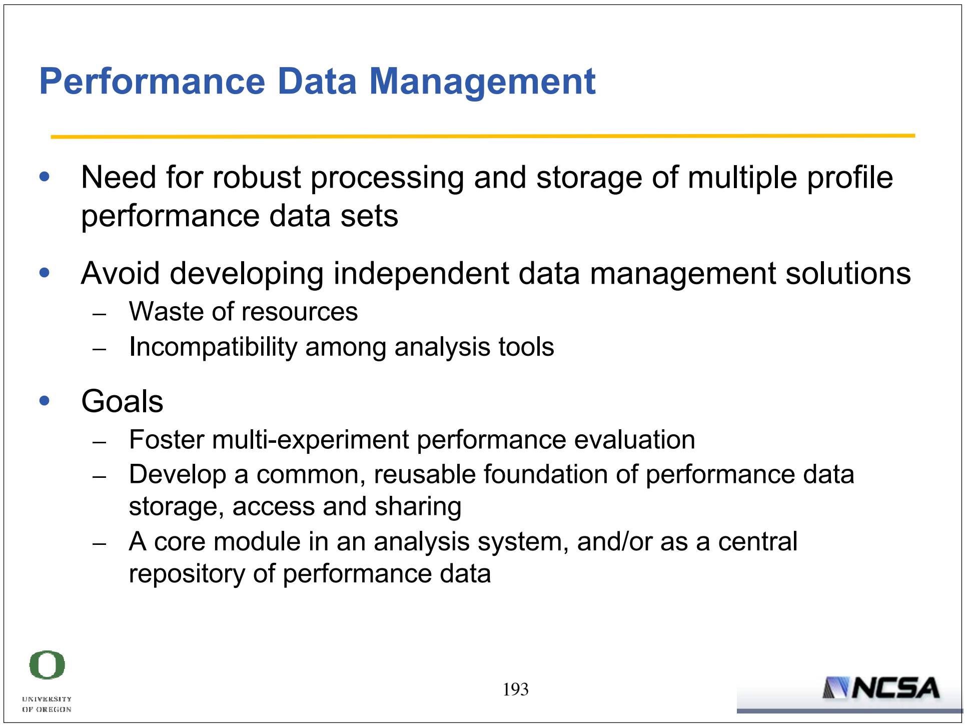
- Follow progression of various displays through time



T = 0s

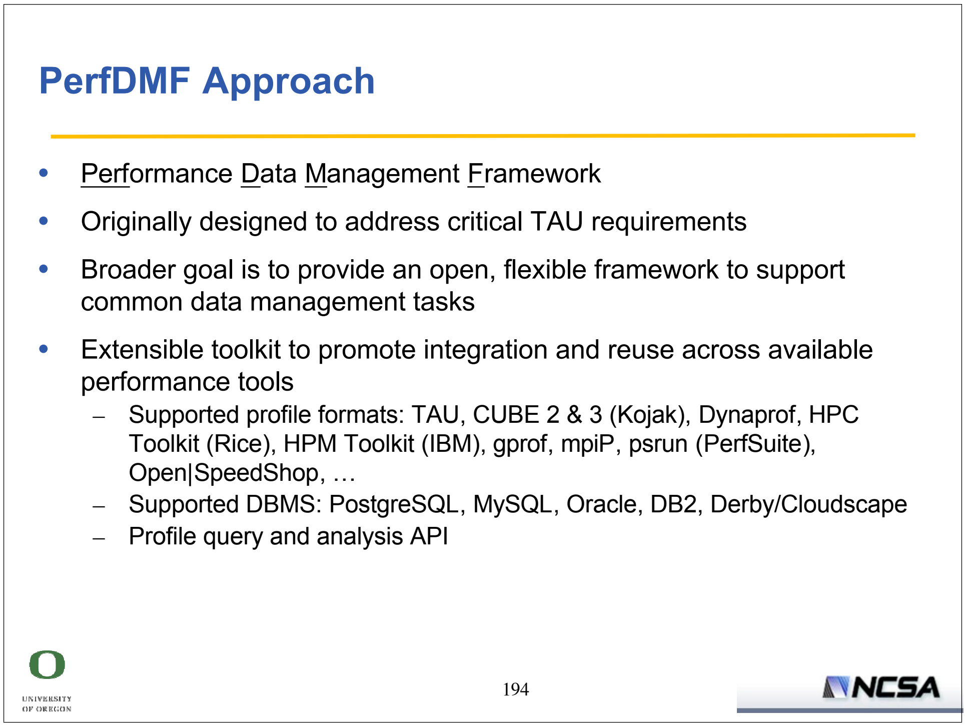
T = 11s

TAU: ParaProf: 3D Visualizer: ../snapshot256.xml
File Options Windows Help
0.177
0.097
bits (Exclusive, Linux Timers)
MPI_Init() (Exclusive, Linux Timers)
rhs (Exclusive, Linux Timers)
MPI_Recv() (Exclusive, Linux Timers)
2.188
1.9
1.611
1.323
1.035
TAU: ParaProf: 3D Visualizer: ../snapshot256.xml
File Options Windows Help
0.775
0.429
bits (Exclusive, Linux Timers)
MPI_Init() (Exclusive, Linux Timers)
rhs (Exclusive, Linux Timers)
MPI_Recv() (Exclusive, Linux Timers)



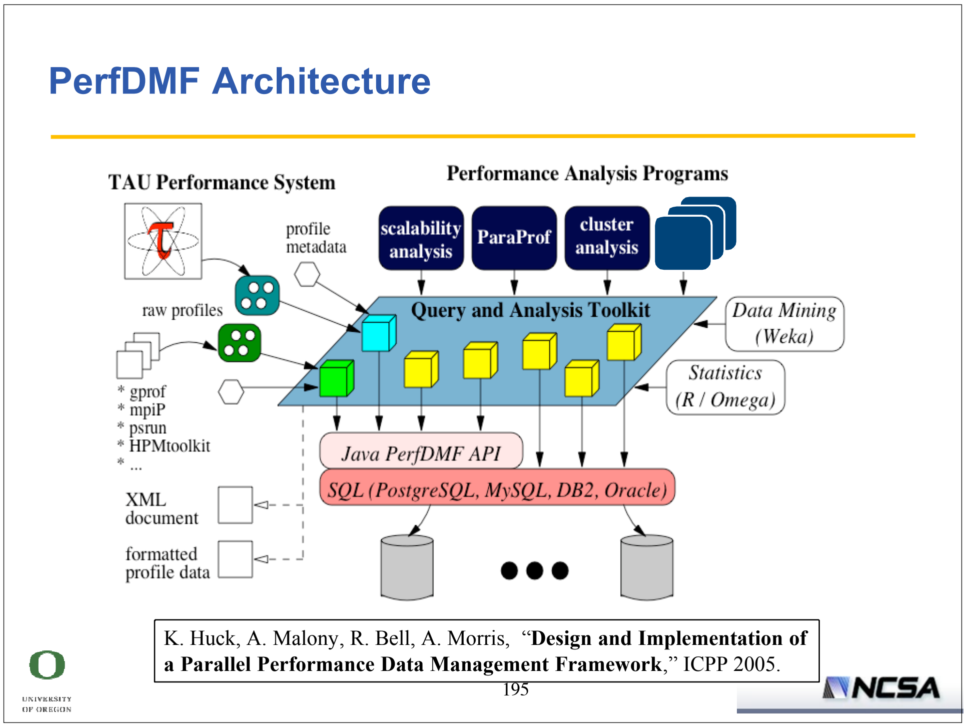
Performance Data Management

- Need for robust processing and storage of multiple profile performance data sets
- Avoid developing independent data management solutions
  - Waste of resources
  - Incompatibility among analysis tools
- Goals
  - Foster multi-experiment performance evaluation
  - Develop a common, reusable foundation of performance data storage, access and sharing
  - A core module in an analysis system, and/or as a central repository of performance data



PerfDMF Approach

- Performance Data Management Framework
- Originally designed to address critical TAU requirements
- Broader goal is to provide an open, flexible framework to support common data management tasks
- Extensible toolkit to promote integration and reuse across available performance tools
  - Supported profile formats: TAU, CUBE 2 & 3 (Kojak), Dynaprof, HPC Toolkit (Rice), HPM Toolkit (IBM), gprof, mpiP, psrun (PerfSuite), Open|SpeedShop, …
  - Supported DBMS: PostgreSQL, MySQL, Oracle, DB2, Derby/Cloudscape
  - Profile query and analysis API



PerfDMF Architecture

TAU Performance System

Performance Analysis Programs

scalability analysis

ParaProf

cluster analysis

profile metadata

raw profiles

Query and Analysis Toolkit

Data Mining (Weka)

Statistics (R / Omega)

- gprof
- mpiP
- psrun
- HPMtoolkit
- ...

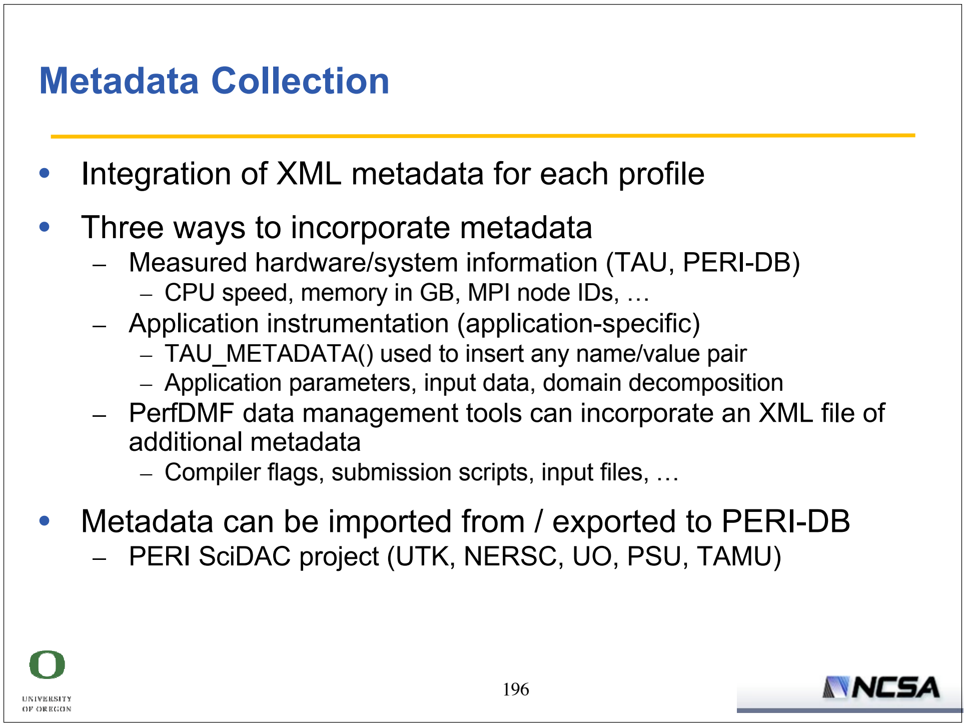
Java PerfDMF API

SQL (PostgreSQL, MySQL, DB2, Oracle)

XML document

formatted profile data

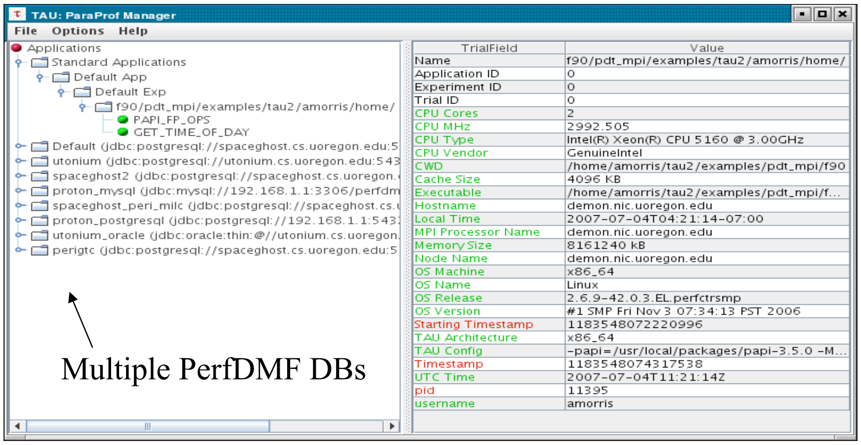
K. Huck, A. Malony, R. Bell, A. Morris, "**Design and Implementation of a Parallel Performance Data Management Framework**," ICPP 2005.



Metadata Collection

- Integration of XML metadata for each profile
- Three ways to incorporate metadata
  - Measured hardware/system information (TAU, PERI-DB)
    - CPU speed, memory in GB, MPI node IDs, …
  - Application instrumentation (application-specific)
    - TAU_METADATA() used to insert any name/value pair
    - Application parameters, input data, domain decomposition
  - PerfDMF data management tools can incorporate an XML file of additional metadata
    - Compiler flags, submission scripts, input files, …
- Metadata can be imported from / exported to PERI-DB
  - PERI SciDAC project (UTK, NERSC, UO, PSU, TAMU)

Metadata for Each Experiment



- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - f90/pdt_mpi/examples/tau2/amorris/home/
          - PAPI_FP_OPS
          - GET_TIME_OF_DAY
  - Default (jdbc:postgresql://spaceghost.cs.uoregon.edu:5
  - utonium (jdbc:postgresql://utonium.cs.uoregon.edu:543
  - spaceghost2 (jdbc:postgresql://spaceghost.cs.uoregon.
  - proton_mysql (jdbc:mysql://192.168.1.1:3306/perfdm
  - spaceghost_peri_milc (jdbc:postgresql://spaceghost.cs.u
  - proton_postgresql (jdbc:postgresql://192.168.1.1:543
  - utonium_oracle (jdbc:oracle:thin:@//utonium.cs.uoregon.
  - perigtc (jdbc:postgresql://spaceghost.cs.uoregon.edu:5

TrialField	Value
Name	f90/pdt_mpi/examples/tau2/amorris/home/
Application ID	0
Experiment ID	0
Trial ID	0
CPU Cores	2
CPU MHz	2992.505
CPU Type	Intel(R) Xeon(R) CPU 5160 @ 3.00GHz
CPU Vendor	GenuineIntel
CWD	/home/amorris/tau2/examples/pdt_mpi/f90
Cache Size	4096 KB
Executable	/home/amorris/tau2/examples/pdt_mpi/f...
Hostname	demon.nic.uoregon.edu
Local Time	2007-07-04T04:21:14-07:00
MPI Processor Name	demon.nic.uoregon.edu
Memory Size	8161240 kB
Node Name	demon.nic.uoregon.edu
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.9-42.0.3.EL.perfctrsmp
OS Version	#1 SMP Fri Nov 3 07:34:13 PST 2006
Starting Timestamp	1183548072220996
TAU Architecture	x86_64
TAU Config	-papi=/usr/local/packages/papi-3.5.0 -M...
Timestamp	1183548074317538
UTC Time	2007-07-04T11:21:14Z
pid	11395
username	amorris

Multiple PerfDMF DBs

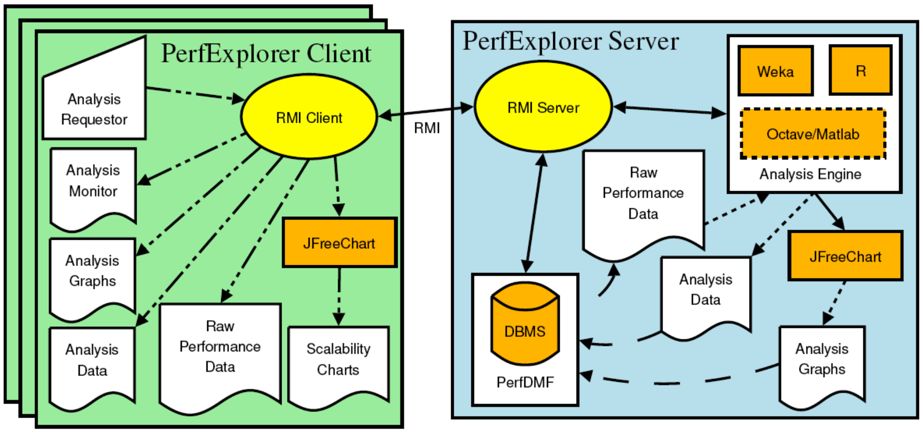
Performance Data Mining

- Conduct parallel performance analysis process
  - In a systematic, collaborative and reusable manner
  - Manage performance complexity
  - Discover performance relationship and properties
  - Automate process
- Multi-experiment performance analysis
- Large-scale performance data reduction
  - Summarize characteristics of large processor runs
- Implement extensible analysis framework
  - Abstraction / automation of data mining operations
  - Interface to existing analysis and data mining tools

Performance Data Mining (PerfExplorer)

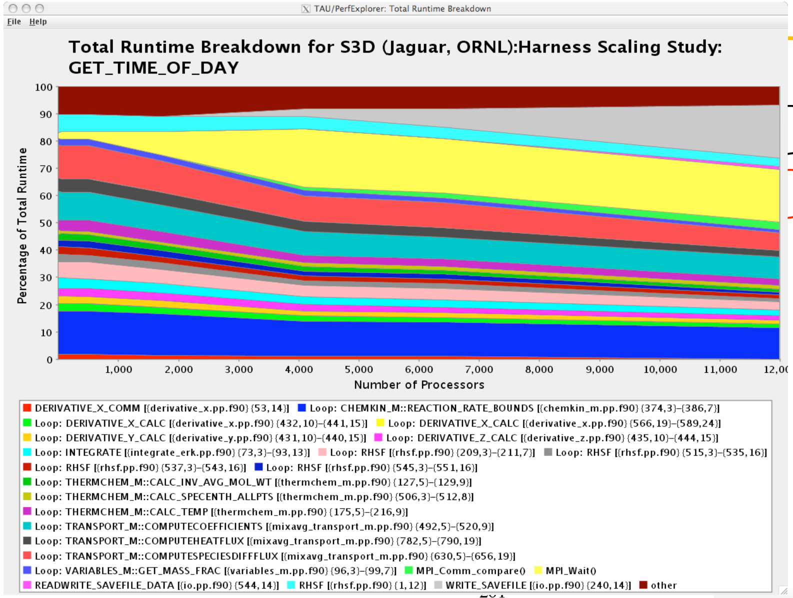
- Performance knowledge discovery framework
  - Data mining analysis applied to parallel performance data
    - comparative, clustering, correlation, dimension reduction, …
  - Use the existing TAU infrastructure
    - TAU performance profiles, PerfDMF
- Technology integration
  - Java API and toolkit for portability
  - Built on top of PerfDMF
  - R-project/Omegahat, Octave/Matlab statistical analysis
  - WEKA data mining package
  - JFreeChart for visualization, vector output (EPS, SVG)

Performance Data Mining (PerfExplorer v1)



K. Huck and A. Malony, "**PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing**," SC 2005.

PerfExplorer: S3D Total Runtime Breakdown



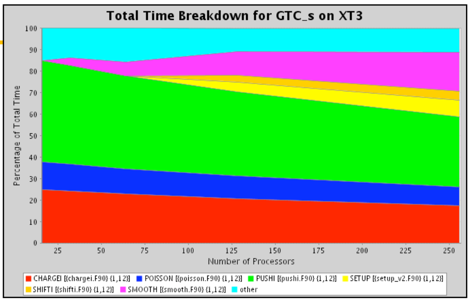
WRITE_ SAVEFILE

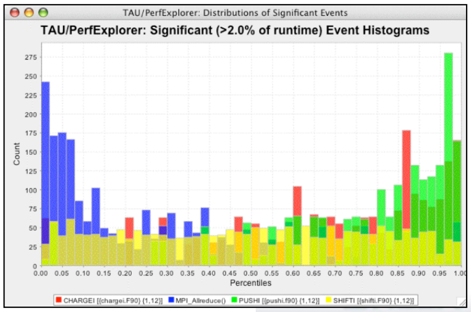
MPI_Wait

12,000 cores!

Relative Comparisons (GTC, XT3, DOE PERI)

- Total execution time
- Timesteps per second
- Relative efficiency
- Relative efficiency per event
- Relative speedup
- Relative speedup per event
- Group fraction of total
- Runtime breakdown
- Correlate events with total runtime
- Relative efficiency per phase
- Relative speedup per phase
- Distribution visualizations

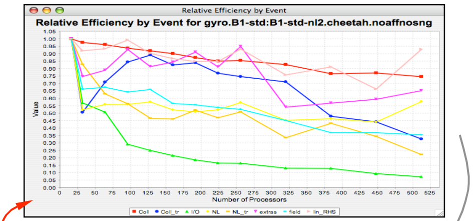


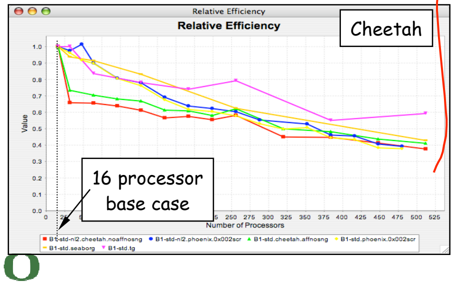


Data: GYRO on various architectures

PerfExplorer – GYRO Relative Efficiency

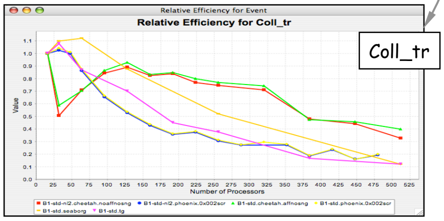
- By experiment (B1-std)
  - Total runtime (Cheetah (red))
- By event for one experiment
  - Coll_tr (blue) is significant
- By experiment for one event
  - Shows how Coll_tr behaves for all experiments
  - Data generated by Pat Worley, ORNL





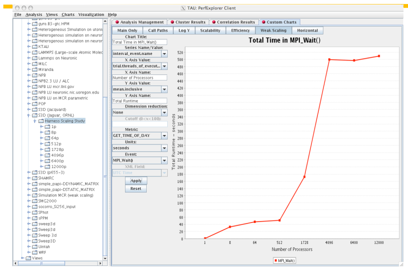
Cheetah

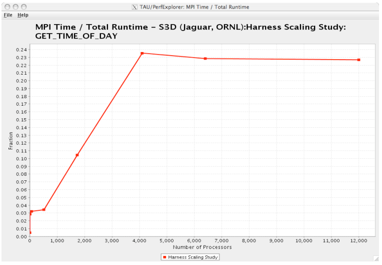
16 processor base case

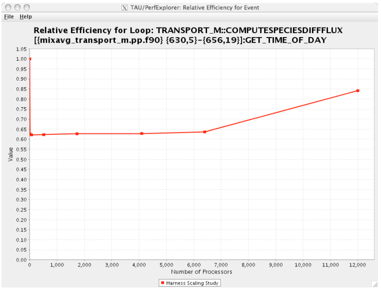


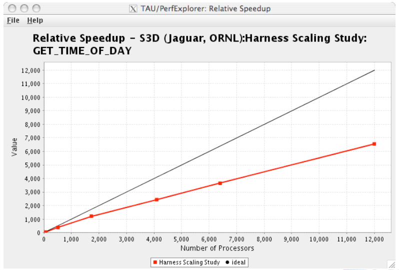
Coll_tr

PerfExplorer: Cross Experiment Analysis for S3D

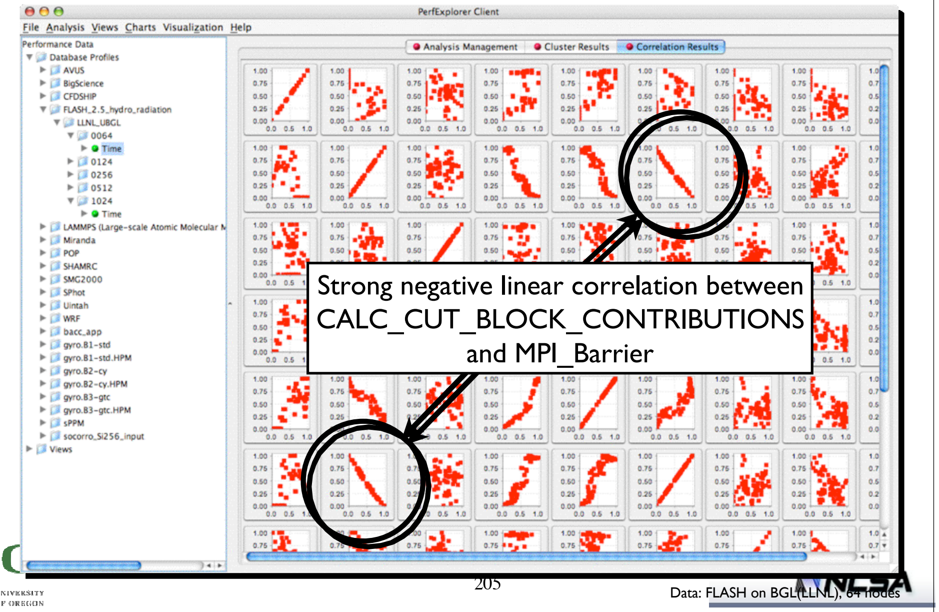








Correlation Analysis

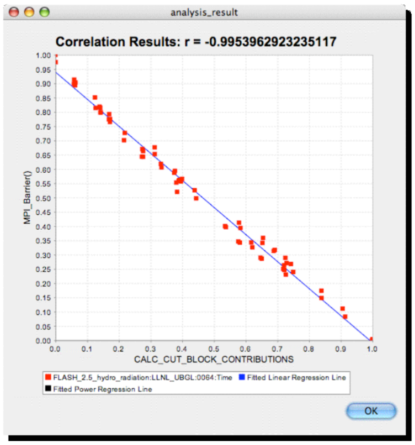


Strong negative linear correlation between CALC_CUT_BLOCK_CONTRIBUTIONS and MPI_Barrier

Data: FLASH on BGL(LLNL), 64 nodes

PerfExplorer - Correlation Analysis (Flash)

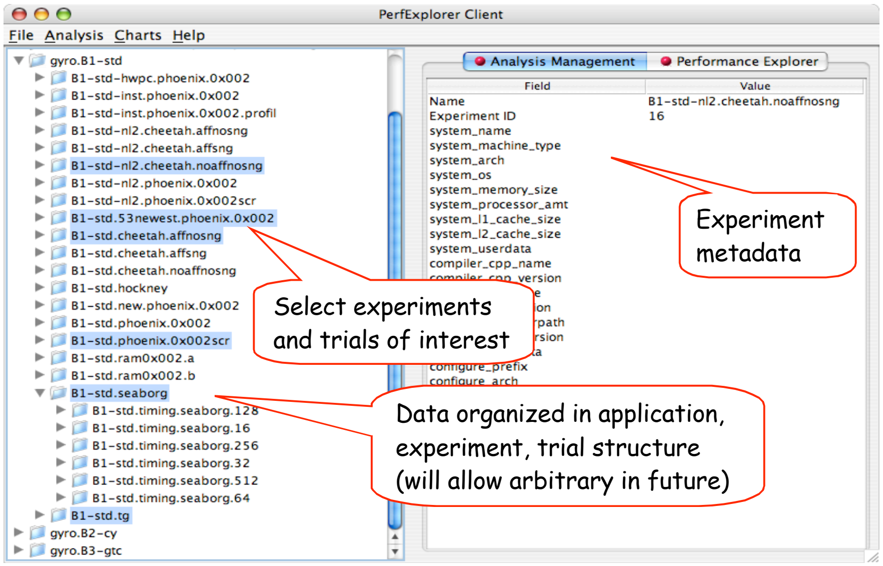
- -0.995 indicates strong, negative relationship
- As CALC_CUT_BLOCK_CONTRIBUTIONS() increases in execution time, MPI_Barrier() decreases



PerfExplorer - Comparative Analysis

- Relative speedup, efficiency
  - total runtime, by event, one event, by phase
- Breakdown of total runtime
- Group fraction of total runtime
- Correlating events to total runtime
- Timesteps per second
- Performance Evaluation Research Center (PERC)
  - PERC tools study (led by ORNL, Pat Worley)
  - In-depth performance analysis of select applications
  - Evaluation performance analysis requirements
  - Test tool functionality and ease of use

PerfExplorer - Interface



- gyro.B1-std
  - B1-std-hwpc.phoenix.0x002
  - B1-std-inst.phoenix.0x002
  - B1-std-inst.phoenix.0x002.profil
  - B1-std-nl2.cheetah.affnosng
  - B1-std-nl2.cheetah.affsng
  - B1-std-nl2.cheetah.noaffnosng
  - B1-std-nl2.phoenix.0x002
  - B1-std-nl2.phoenix.0x002scr
  - B1-std.53newest.phoenix.0x002
  - B1-std.cheetah.affnosng
  - B1-std.cheetah.affsng
  - B1-std.cheetah.noaffnosng
  - B1-std.hockney
  - B1-std.new.phoenix.0x002
  - B1-std.phoenix.0x002
  - B1-std.phoenix.0x002scr
  - B1-std.ram0x002.a
  - B1-std.ram0x002.b
  - B1-std.seaborg
    - B1-std.timing.seaborg.128
    - B1-std.timing.seaborg.16
    - B1-std.timing.seaborg.256
    - B1-std.timing.seaborg.32
    - B1-std.timing.seaborg.512
    - B1-std.timing.seaborg.64
  - B1-std.tg
- gyro.B2-cy
- gyro.B3-gtc

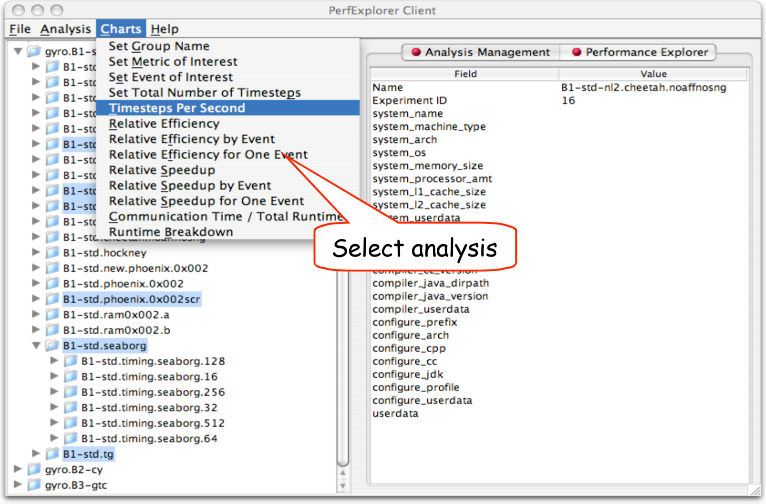
Field	Value
Name	B1-std-nl2.cheetah.noaffnosng
Experiment ID	16
system_name	
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
system_userdata	
compiler_cpp_name	
compiler_cpp_version	
e	
on	
rpath	
rsion	
a	
configure_prefix	
configure_arch	

Experiment metadata

Select experiments and trials of interest

Data organized in application, experiment, trial structure (will allow arbitrary in future)

PerfExplorer - Interface



PerfExplorer Client
File Analysis Charts Help
Set Group Name
Set Metric of Interest
Set Event of Interest
Set Total Number of Timesteps
Timesteps Per Second
Relative Efficiency
Relative Efficiency by Event
Relative Efficiency for One Event
Relative Speedup
Relative Speedup by Event
Relative Speedup for One Event
Communication Time / Total Runtime
Runtime Breakdown
Analysis Management
Performance Explorer

Select analysis

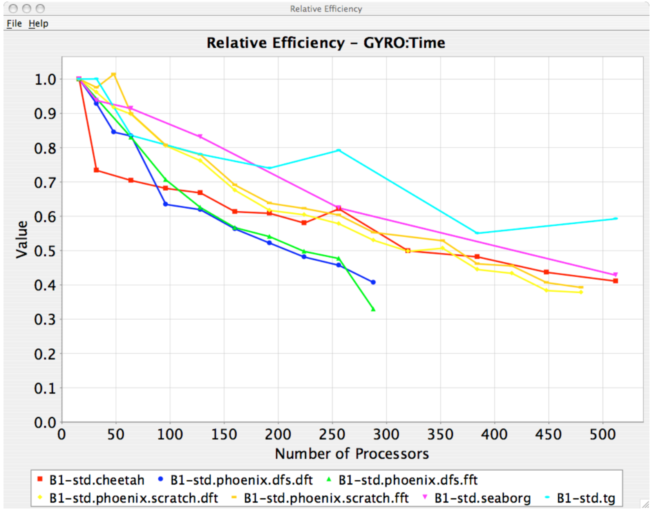
```
▼ gyro.B1-s
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std
  ▶ B1-std.hockney
  ▶ B1-std.new.phoenix.0x002
  ▶ B1-std.phoenix.0x002
  ▶ B1-std.phoenix.0x002scr
  ▶ B1-std.ram0x002.a
  ▶ B1-std.ram0x002.b
  ▼ B1-std.seaborg
    ▶ B1-std.timing.seaborg.128
    ▶ B1-std.timing.seaborg.16
    ▶ B1-std.timing.seaborg.256
    ▶ B1-std.timing.seaborg.32
    ▶ B1-std.timing.seaborg.512
    ▶ B1-std.timing.seaborg.64
  ▶ B1-std.tg
▶ gyro.B2-cy
▶ gyro.B3-gtc
```

Field	Value
Name	B1-std-nl2.cheetah.noaffnosng
Experiment ID	16
system_name	
system_machine_type	
system_arch	
system_os	
system_memory_size	
system_processor_amt	
system_l1_cache_size	
system_l2_cache_size	
...tem_userdata	
compiler_java_dirpath	
compiler_java_version	
compiler_userdata	
configure_prefix	
configure_arch	
configure_cpp	
configure_cc	
configure_jdk	
configure_profile	
configure_userdata	
userdata	

UNIVERSITY OF OREGON

NCSA

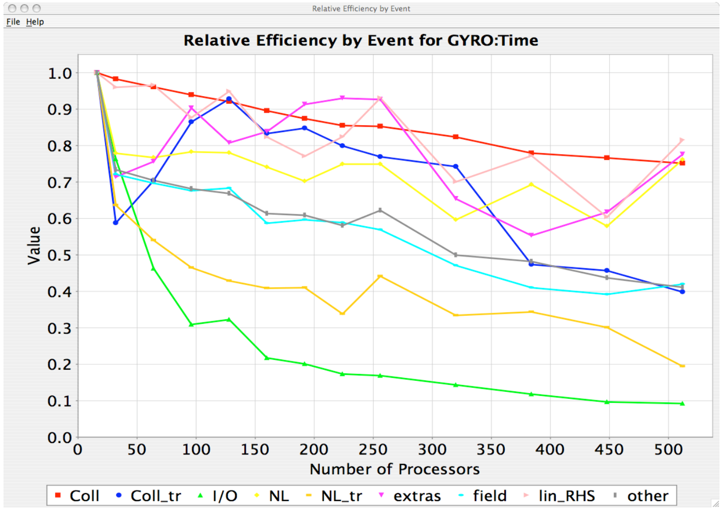
PerfExplorer - Relative Efficiency Plots



UNIVERSITY OF OREGON

NCSA

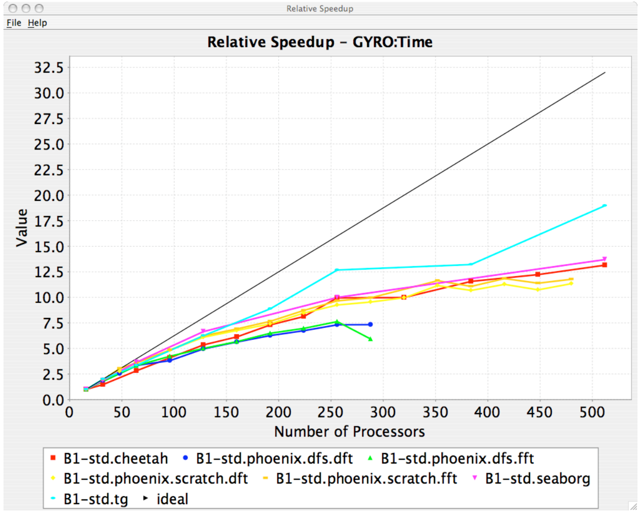
PerfExplorer - Relative Efficiency by Routine



UNIVERSITY OF OREGON

NCSA

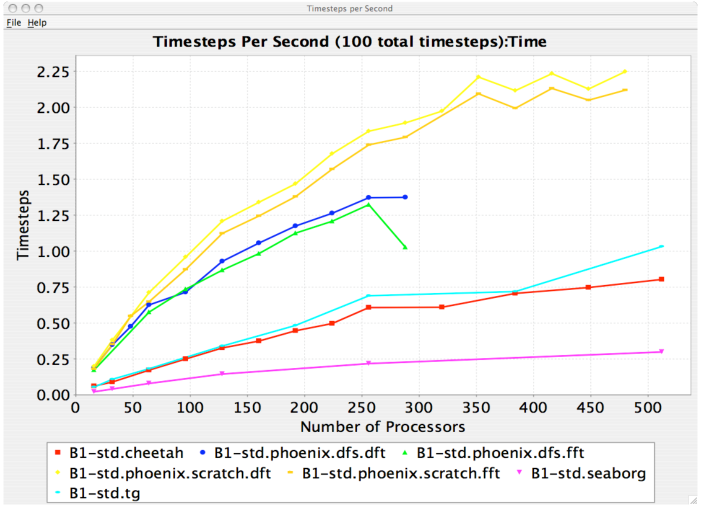
PerfExplorer - Relative Speedup



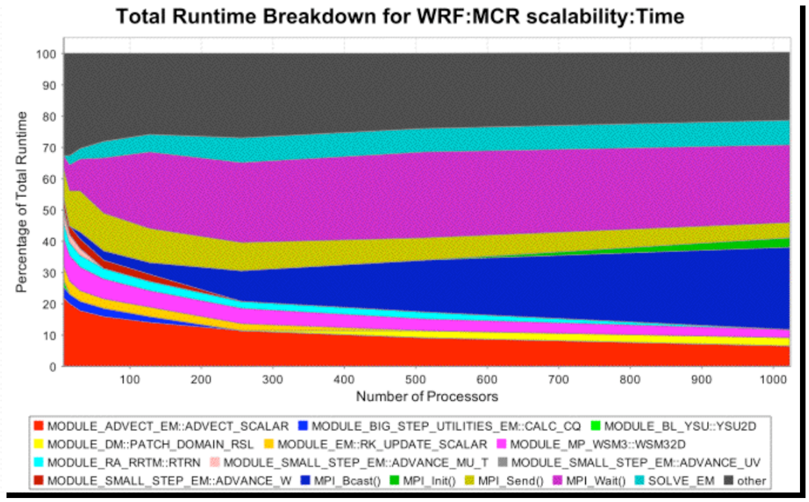
UNIVERSITY OF OREGON

NCSA

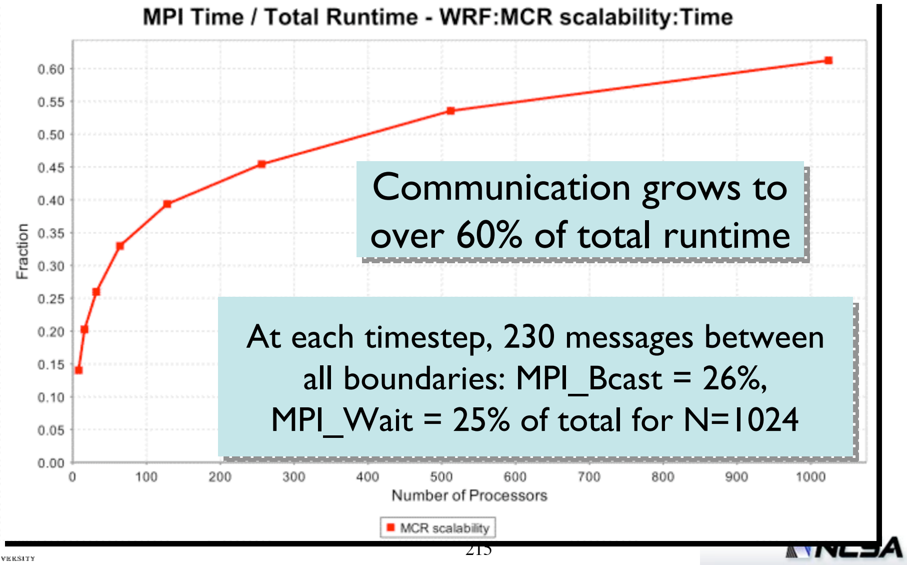
PerfExplorer - Timesteps Per Second



PerfExplorer - Runtime Breakdown



Group % of Total



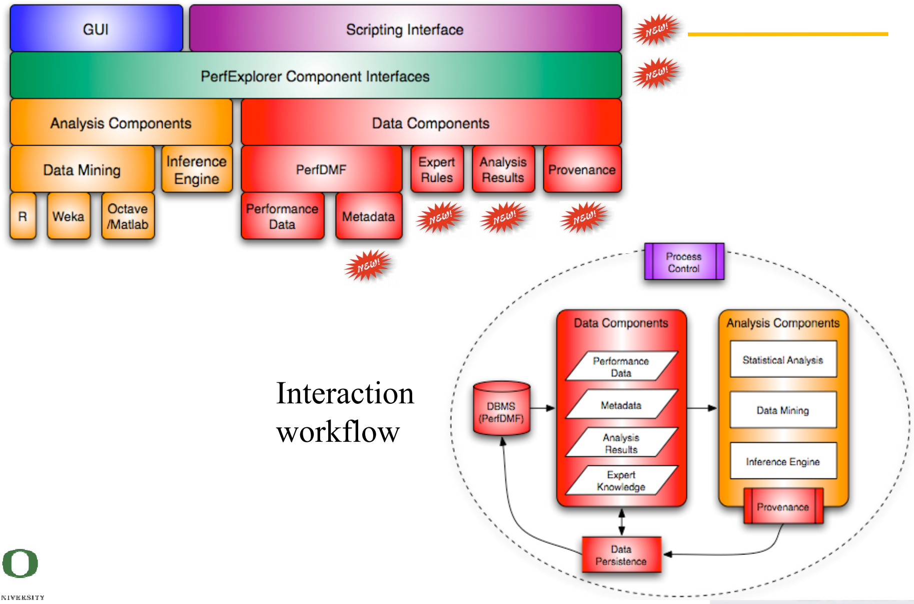
Communication grows to over 60% of total runtime

At each timestep, 230 messages between all boundaries: MPI_Bcast = 26%, MPI_Wait = 25% of total for N=1024

PerfExplorer v2 – Requirements and Features

- Component-based analysis process
  - Analysis operations implemented as modules
  - Linked together in analysis process and workflow
- Scripting
  - Provides process/workflow development and automation
- Metadata input, management, and access
- Inference engine
  - Reasoning about causes of performance phenomena
  - Analysis knowledge captured in expert rules
- Persistence of intermediate results
- Provenance
  - Provides historical record of analysis results

PerfExplorer v2: Architecture and Interaction



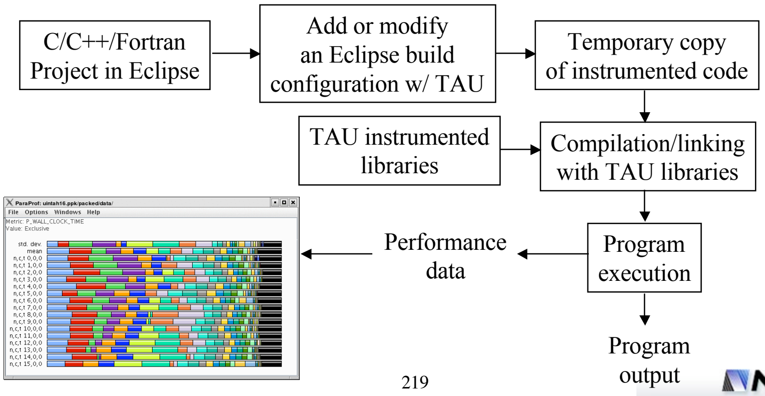
Interaction workflow

TAU Integration with IDEs

- High performance software development environments
  - Tools may be complicated to use
  - Interfaces and mechanisms differ between platforms / OS
- Integrated development environments
  - Consistent development environment
  - Numerous enhancements to development process
  - Standard in industrial software development
- Integrated performance analysis
  - Tools limited to single platform or programming language
  - Rarely compatible with 3rd party analysis tools
  - Little or no support for parallel projects

TAU and Eclipse

- Provide an interface for configuring TAU’s automatic instrumentation within Eclipse’s build system
- Manage runtime configuration settings and environment variables for execution of TAU instrumented programs



C/C++/Fortran Project in Eclipse

Add or modify an Eclipse build configuration w/ TAU

Temporary copy of instrumented code

TAU instrumented libraries

Compilation/linking with TAU libraries

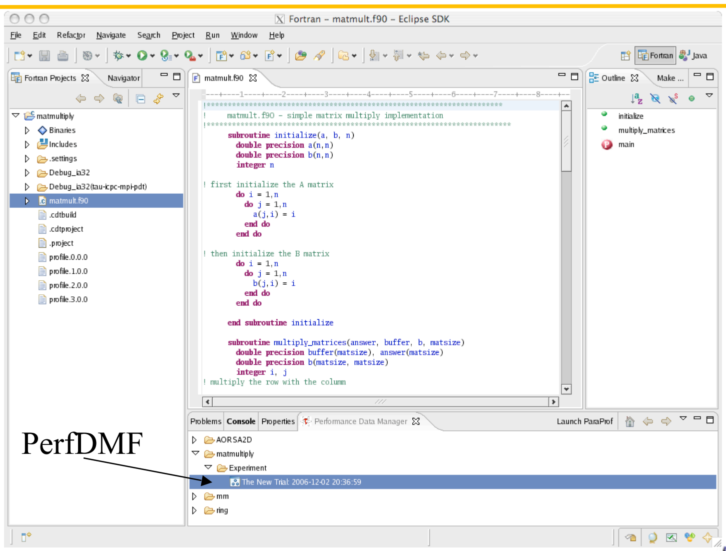
Program execution

Performance data

Program output

ParaProf: uintah16.ppk/packed/data/
File Options Windows Help
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
std. dev.
mean
n,c,t 0,0,0
n,c,t 15,0,0

TAU and Eclipse

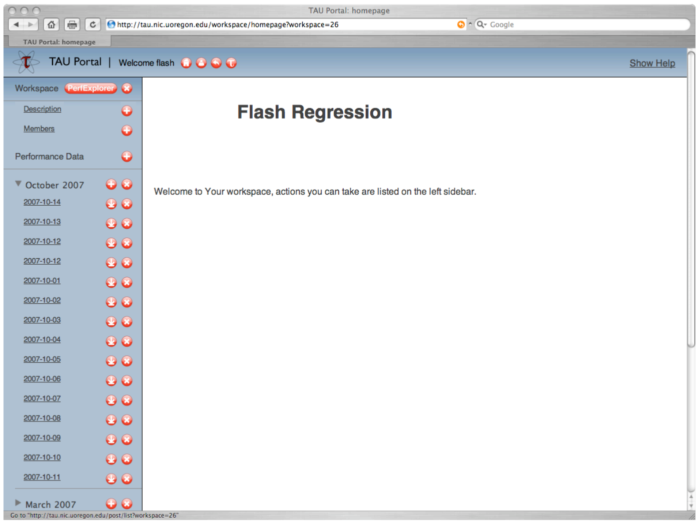


```
!************************************************************************
!     matmult.f90 - simple matrix multiply implementation
!*************************************************************************
      subroutine initialize(a, b, n)
        double precision a(n,n)
        double precision b(n,n)
        integer n

! first initialize the A matrix
        do i = 1,n
          do j = 1,n
            a(j,i) = i
          end do
        end do

! then initialize the B matrix
        do i = 1,n
          do j = 1,n
            b(j,i) = i
          end do
        end do

      end subroutine initialize

      subroutine multiply_matrices(answer, buffer, b, matsize)
        double precision buffer(matsize), answer(matsize)
        double precision b(matsize, matsize)
        integer i, j
! multiply the row with the column
```

TAU Portal

- Web-based access to TAU
- Support collaborative performance study
  - Secure performance data sharing
  - Does not require TAU installation
  - Launch TAU performance tools with Java WebStart
    - ParaProf, PerfExplorer
- FLASH regression testing
  - Nightly regression testcases
  - Uploaded to the database automatically
  - Interactive review of performance through TAU portal
  - Multi-experiment analysis

Portal: Nightly Performance Regression Testing

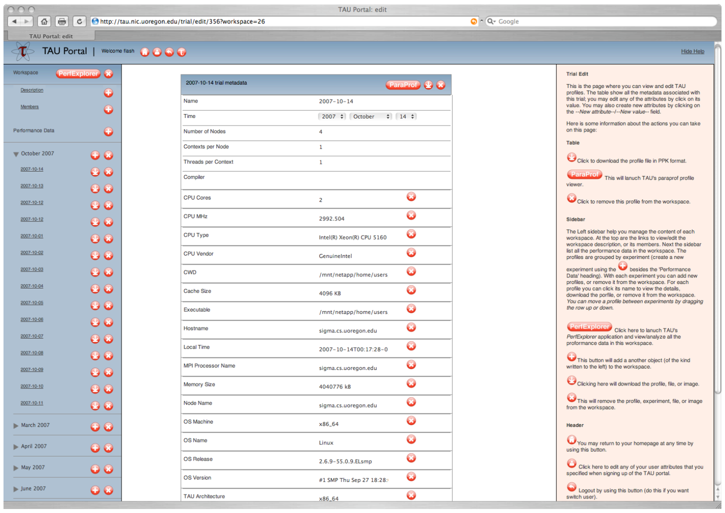


Flash Regression

Welcome to Your workspace, actions you can take are listed on the left sidebar.

Go to "http://tau.nic.uoregon.edu/post/list?workspace=26"

TAU Portal: Launch ParaProf/PerfExplorer



2007-10-14 trial metadata

Name	2007-10-14
Time	2007 October 14
Number of Nodes	4
Contexts per Node	1
Threads per Context	1
Compiler	
CPU Cores	2
CPU MHz	2992.504
CPU Type	Intel(R) Xeon(R) CPU 5160
CPU Vendor	GenuineIntel
CWD	/mnt/netapp/home/users
Cache Size	4096 KB
Executable	/mnt/netapp/home/users
Hostname	sigma.cs.uoregon.edu
Local Time	2007-10-14T00:17:28-0
MPI Processor Name	sigma.cs.uoregon.edu
Memory Size	4040776 kB
Node Name	sigma.cs.uoregon.edu
OS Machine	x86_64
OS Name	Linux
OS Release	2.6.9-55.0.9.ELsmp
OS Version	#1 SMP Thu Sep 27 18:28:
TAU Architecture	x86_64

Trial Edit

This is the page where you can view and edit TAU profiles. The table show all the metadata associated with this trial; you may edit any of the attributes by click on its value. You may also create new attributes by clicking on the *--New attribute--*/*--New value--* field.

Here is some information about the actions you can take on this page:

Table

Click to download the profile file in PPK format.

ParaProf This will lanuch TAU's paraprof profile viewer.

Click to remove this profile from the workspace.

Sidebar

The Left sidebar help you manage the content of each workspace. At the top are the links to view/edit the workspace description, or its members. Next the sidebar list all the performance data in the workspace. The profiles are grouped by experiment (create a new experiment using the ⊕ besides the 'Performance Data' heading). With each experiment you can add new profiles, or remove it from the workspace. For each profile you can click its name to view the details, download the porfile, or remove it from the workspace. *You can move a profile between experiments by dragging the row up or down.*

PerfExplorer Click here to lanuch TAU's *PerfExplorer* application and view/analyze all the proformance data in this workspace.

This button will add a another object (of the kind written to the left) to the workspace.

Clicking here will download the profile, file, or image.

This will remove the profile, experiment, file, or image from the workspace.

Header

You may return to your homepage at any time by using this button.

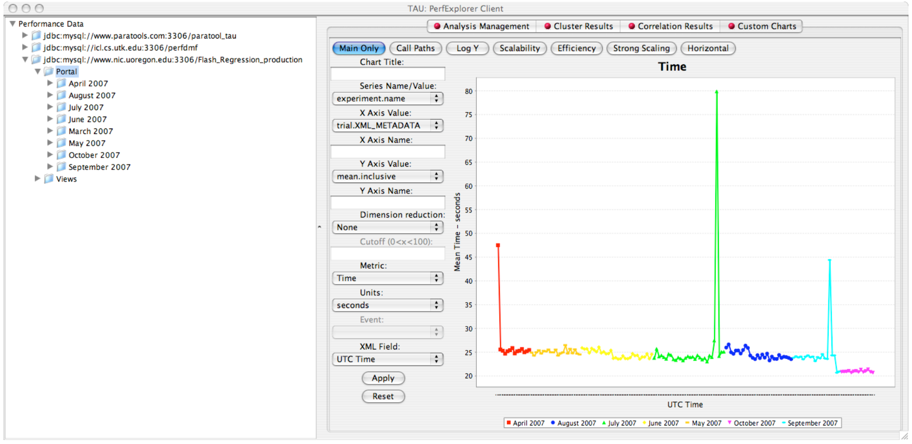
Click here to edit any of your user attributes that you specified when signing up of the TAU portal.

Logout by using this button (do this if you want switch user).

UNIVERSITY OF OREGON

NCSA

PerfExplorer: Regression Testing



Performance Data
- jdbc:mysql://www.paratools.com:3306/paratool_tau
- jdbc:mysql://icl.cs.utk.edu:3306/perfdmf
- jdbc:mysql://www.nic.uoregon.edu:3306/Flash_Regression_production
  - Portal
    - April 2007
    - August 2007
    - July 2007
    - June 2007
    - March 2007
    - May 2007
    - October 2007
    - September 2007
  - Views

Chart Title:

Series Name/Value: experiment.name

X Axis Value: trial.XML_METADATA

X Axis Name:

Y Axis Value: mean.inclusive

Y Axis Name:

Dimension reduction: None

Cutoff (0<x<100):

Metric: Time

Units: seconds

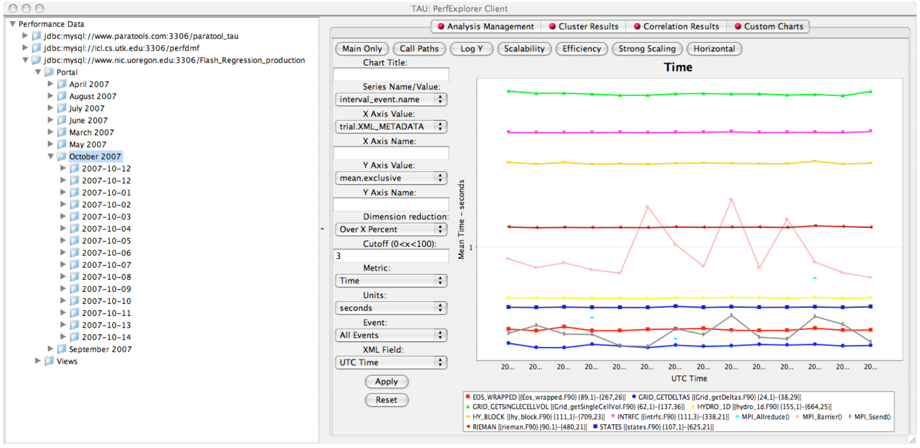
Event:

XML Field: UTC Time

UNIVERSITY OF OREGON

NCSA

PerfExplorer: Limiting Events (> 3%), Oct 2007



TAU: PerfExplorer Client

- Performance Data
  - jdbc:mysql://www.paratools.com:3306/paratool_tau
  - jdbc:mysql://icl.cs.utk.edu:3306/perfdmf
  - jdbc:mysql://www.nic.uoregon.edu:3306/Flash_Regression_production
    - Portal
      - April 2007
      - August 2007
      - July 2007
      - June 2007
      - March 2007
      - May 2007
      - October 2007
        - 2007-10-12
        - 2007-10-12
        - 2007-10-01
        - 2007-10-02
        - 2007-10-03
        - 2007-10-04
        - 2007-10-05
        - 2007-10-06
        - 2007-10-07
        - 2007-10-08
        - 2007-10-09
        - 2007-10-10
        - 2007-10-11
        - 2007-10-13
        - 2007-10-14
      - September 2007
    - Views

Chart Title:

Series Name/Value: interval_event.name

X Axis Value: trial.XML_METADATA

X Axis Name:

Y Axis Value: mean.exclusive

Y Axis Name:

Dimension reduction: Over X Percent

Cutoff (0<x<100): 3

Metric: Time

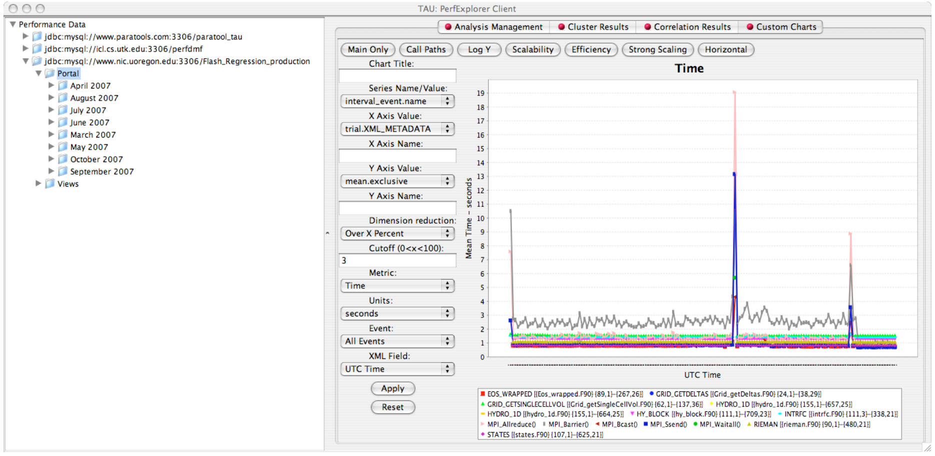
Units: seconds

Event: All Events

XML Field: UTC Time

Analysis Management
Cluster Results
Correlation Results
Custom Charts
Main Only
Call Paths
Log Y
Scalability
Efficiency
Strong Scaling
Horizontal
Apply
Reset
Time
Mean Time - seconds
1
20...
20...
20...
20...
20...
20...
20...
20...
20...
20...
20...
20...
20...
20...
UTC Time
■ EOS_WRAPPED [{Eos_wrapped.F90} {89,1}-{267,26}]
● GRID_GETDELTAS [{Grid_getDeltas.F90} {24,1}-{38,29}]
▲ GRID_GETSINGLECELLVOL [{Grid_getSingleCellVol.F90} {62,1}-{137,36}]
◆ HYDRO_1D [{hydro_1d.F90} {155,1}-{664,25}]
▬ HY_BLOCK [{hy_block.F90} {111,1}-{709,23}]
▼ INTRFC [{intrfc.F90} {111,3}-{338,21}]
▬ MPI_Allreduce()
▶ MPI_Barrier()
▮ MPI_Ssend()
◀ RIEMAN [{rieman.F90} {90,1}-{480,21}]
■ STATES [{states.F90} {107,1}-{625,21}]

PerfExplorer: Exclusive Time for Events (2007)



TAU: PerfExplorer Client

- Performance Data
  - jdbc:mysql://www.paratools.com:3306/paratool_tau
  - jdbc:mysql://icl.cs.utk.edu:3306/perfdmf
  - jdbc:mysql://www.nic.uoregon.edu:3306/Flash_Regression_production
    - Portal
      - April 2007
      - August 2007
      - July 2007
      - June 2007
      - March 2007
      - May 2007
      - October 2007
      - September 2007
    - Views

Chart Title:

Series Name/Value: interval_event.name

X Axis Value: trial.XML_METADATA

X Axis Name:

Y Axis Value: mean.exclusive

Y Axis Name:

Dimension reduction: Over X Percent

Cutoff (0<x<100): 3

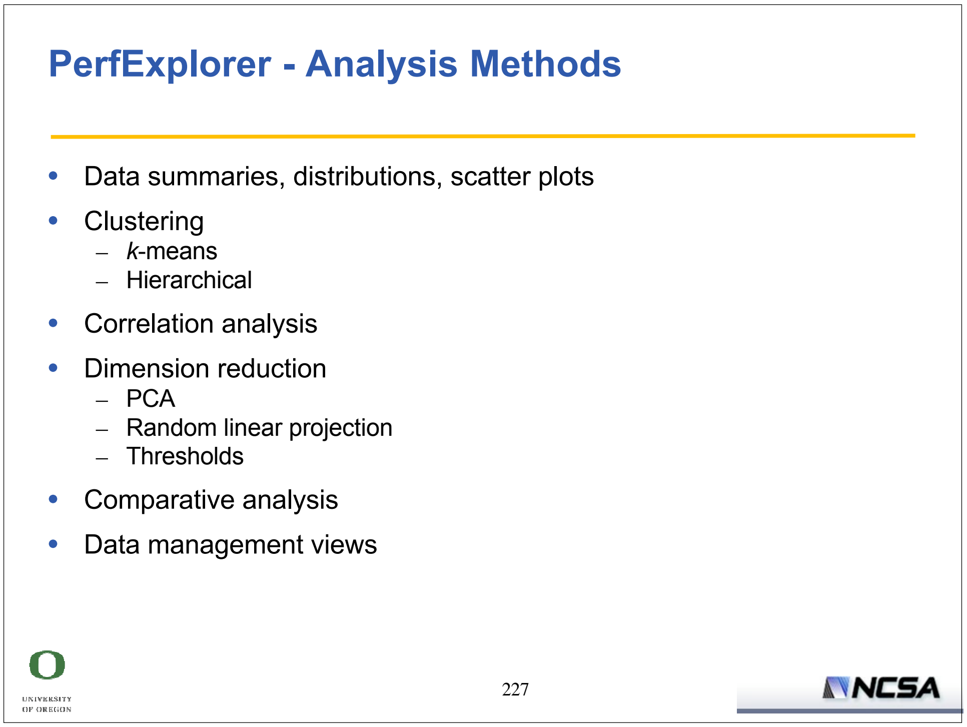
Metric: Time

Units: seconds

Event: All Events

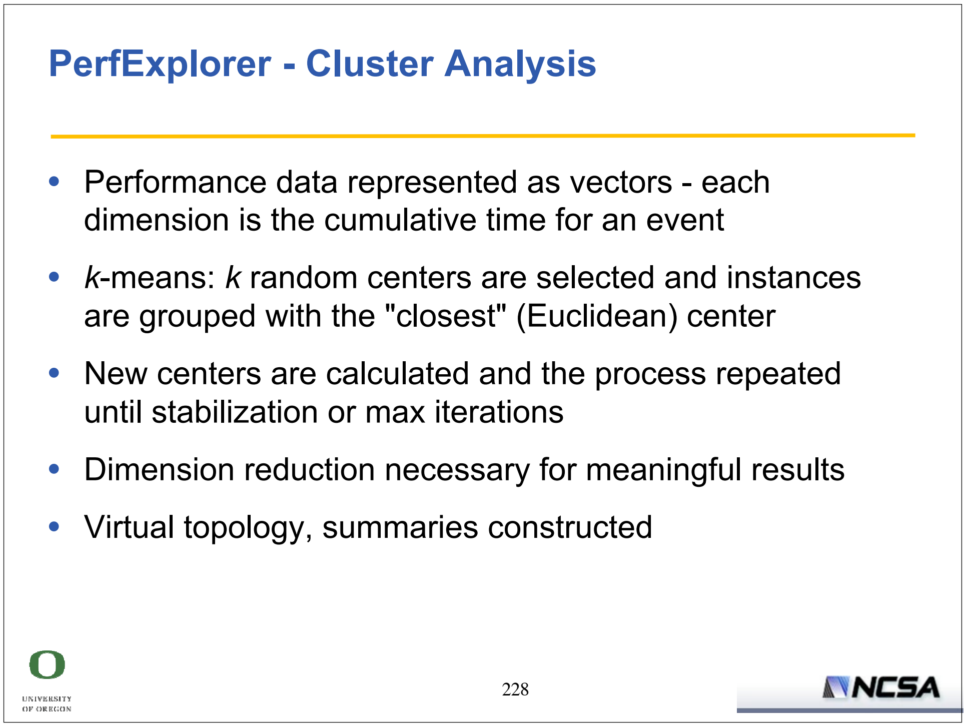
XML Field: UTC Time

Analysis Management
Cluster Results
Correlation Results
Custom Charts
Main Only
Call Paths
Log Y
Scalability
Efficiency
Strong Scaling
Horizontal
Apply
Reset
Time
Mean Time - seconds
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
UTC Time
■ EOS_WRAPPED [{Eos_wrapped.F90} {89,1}-{267,26}]
● GRID_GETDELTAS [{Grid_getDeltas.F90} {24,1}-{38,29}]
▲ GRID_GETSINGLECELLVOL [{Grid_getSingleCellVol.F90} {62,1}-{137,36}]
◆ HYDRO_1D [{hydro_1d.F90} {155,1}-{657,25}]
▬ HYDRO_1D [{hydro_1d.F90} {155,1}-{664,25}]
▼ HY_BLOCK [{hy_block.F90} {111,1}-{709,23}]
▬ INTRFC [{intrfc.F90} {111,3}-{338,21}]
▶ MPI_Allreduce()
▮ MPI_Barrier()
◀ MPI_Bcast()
■ MPI_Ssend()
● MPI_Waitall()
▲ RIEMAN [{rieman.F90} {90,1}-{480,21}]
◆ STATES [{states.F90} {107,1}-{625,21}]



PerfExplorer - Analysis Methods

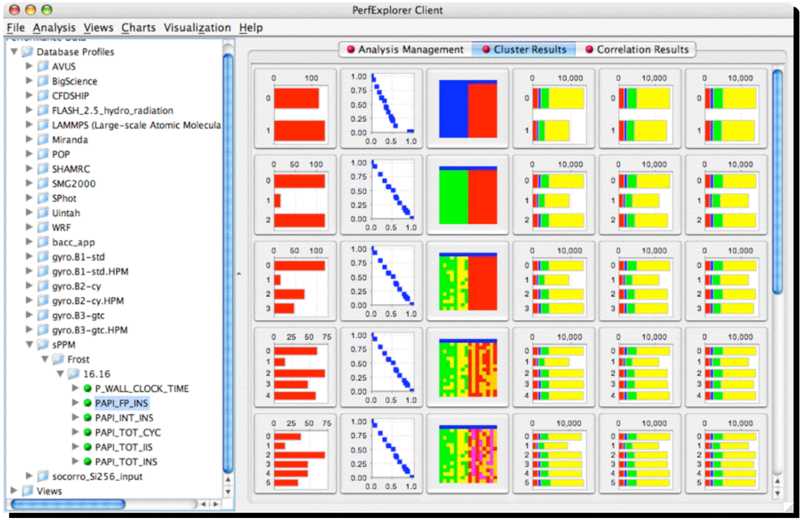
- Data summaries, distributions, scatter plots
- Clustering
  - *k*-means
  - Hierarchical
- Correlation analysis
- Dimension reduction
  - PCA
  - Random linear projection
  - Thresholds
- Comparative analysis
- Data management views



PerfExplorer - Cluster Analysis

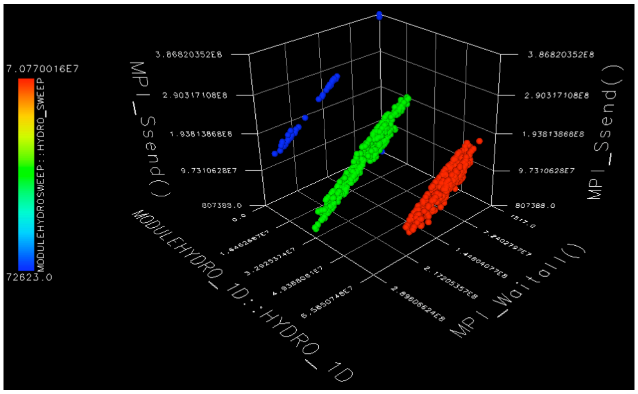
- Performance data represented as vectors - each dimension is the cumulative time for an event
- *k*-means: *k* random centers are selected and instances are grouped with the "closest" (Euclidean) center
- New centers are calculated and the process repeated until stabilization or max iterations
- Dimension reduction necessary for meaningful results
- Virtual topology, summaries constructed

PerfExplorer - Cluster Analysis (sPPM)



PerfExplorer - Cluster Analysis

- Four significant events automatically selected (from 16K processors)
- Clusters and correlations are visible

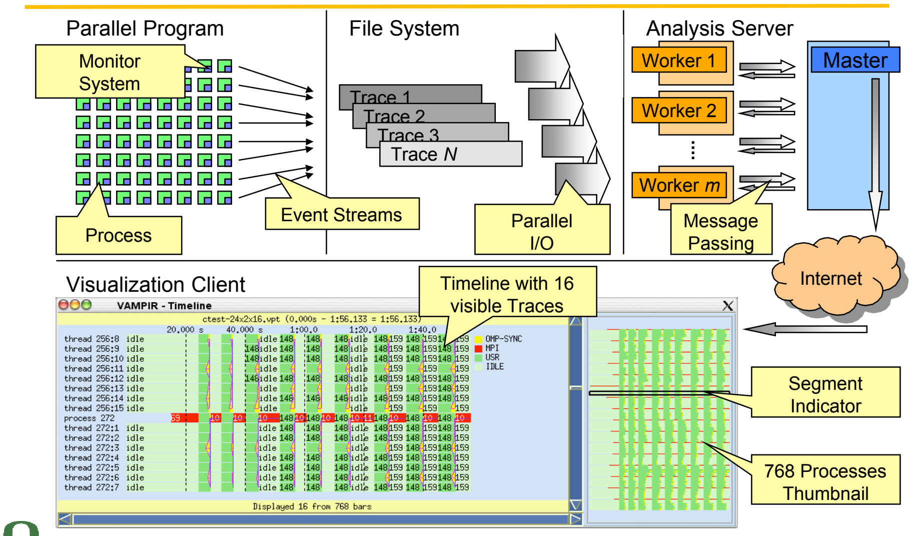


Vampir, VNG, and OTF

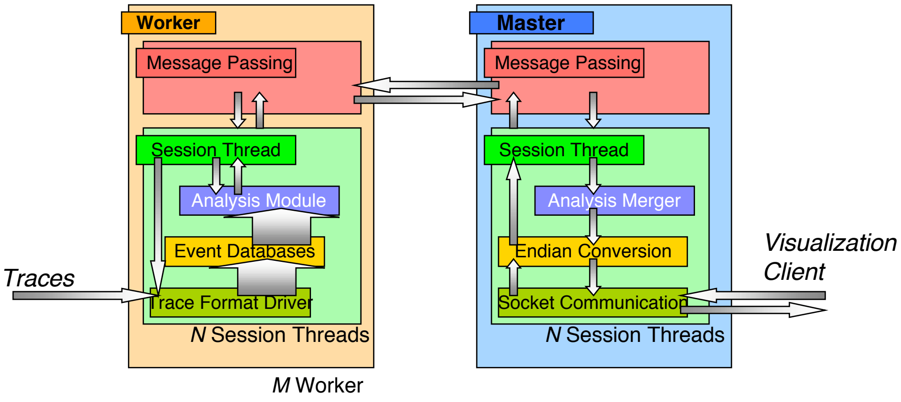
- Commercial trace based tools developed at ZiH, T.U. Dresden
  - Wolfgang Nagel, Holger Brunst and others…
- Vampir Trace Visualizer (aka Intel ® Trace Analyzer v4.0)
  - Sequential program
- Vampir Next Generation (VNG)
  - Client (vng) runs on a desktop, server (vngd) on a cluster
  - Parallel trace analysis
  - Orders of magnitude bigger traces (more memory)
  - State of the art in parallel trace visualization
- Open Trace Format (OTF)
  - Hierarchical trace format, efficient streams based parallel access with VNGD
  - Replacement for proprietary formats such as STF
  - Tracing library available with a evaluation license now. Open source package at SC’06.

http://www.vampir-ng.de

Vampir Next Generation (VNG) Architecture

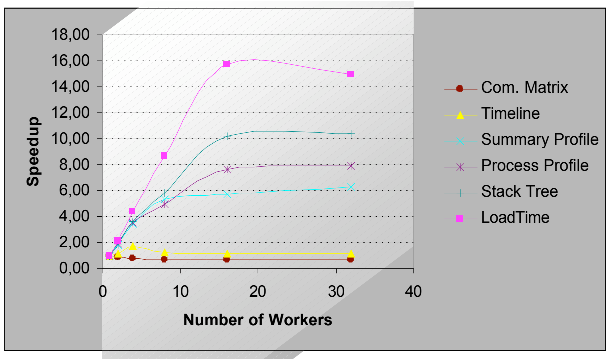


VNG Parallel Analysis Server

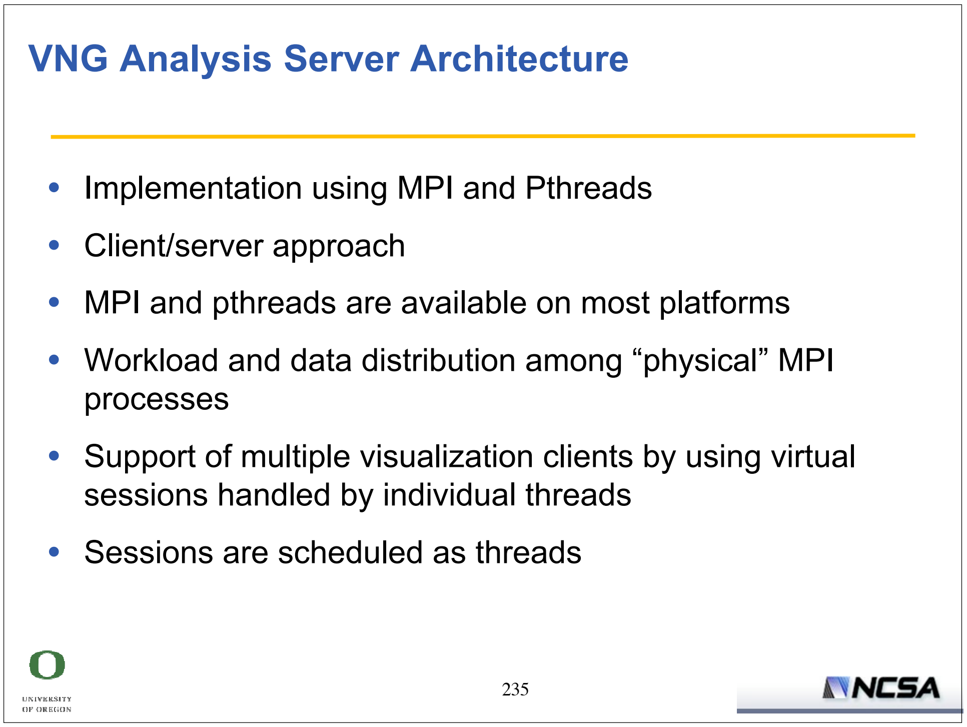


Scalability of VNG [Holger Brunst, WAPA 2005]

- sPPM
- 16 CPUs
- 200 MB

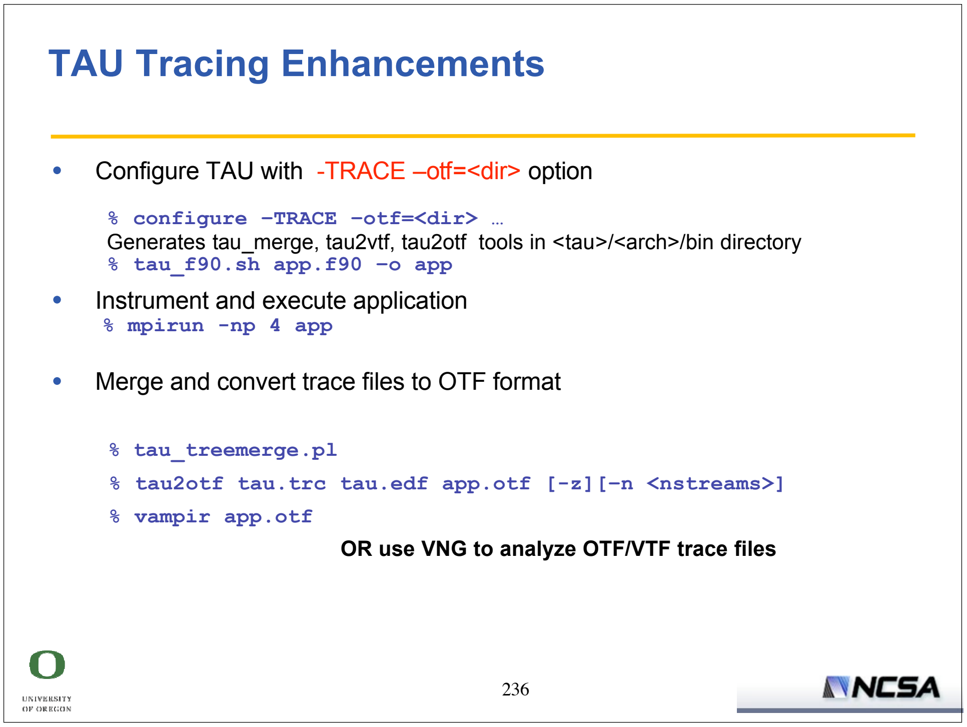


Number of Workers	1	2	4	8	16	32
Load Time	47,33	22,48	10,80	5,43	3,01	3,16
Timeline	0,10	0,09	0,06	0,08	0,09	0,09
Summary Profile	1,59	0,87	0,47	0,30	0,28	0,25
Process Profile	1,32	0,70	0,38	0,26	0,17	0,17
Com. Matrix	0,06	0,07	0,08	0,09	0,09	0,09
Stack Tree	2,57	1,39	0,70	0,44	0,25	0,25



VNG Analysis Server Architecture

- Implementation using MPI and Pthreads
- Client/server approach
- MPI and pthreads are available on most platforms
- Workload and data distribution among “physical” MPI processes
- Support of multiple visualization clients by using virtual sessions handled by individual threads
- Sessions are scheduled as threads



TAU Tracing Enhancements

- Configure TAU with -TRACE –otf=<dir> option

```
% configure -TRACE -otf=<dir> …
```

Generates tau_merge, tau2vtf, tau2otf tools in <tau>/<arch>/bin directory

```
% tau_f90.sh app.f90 -o app
```

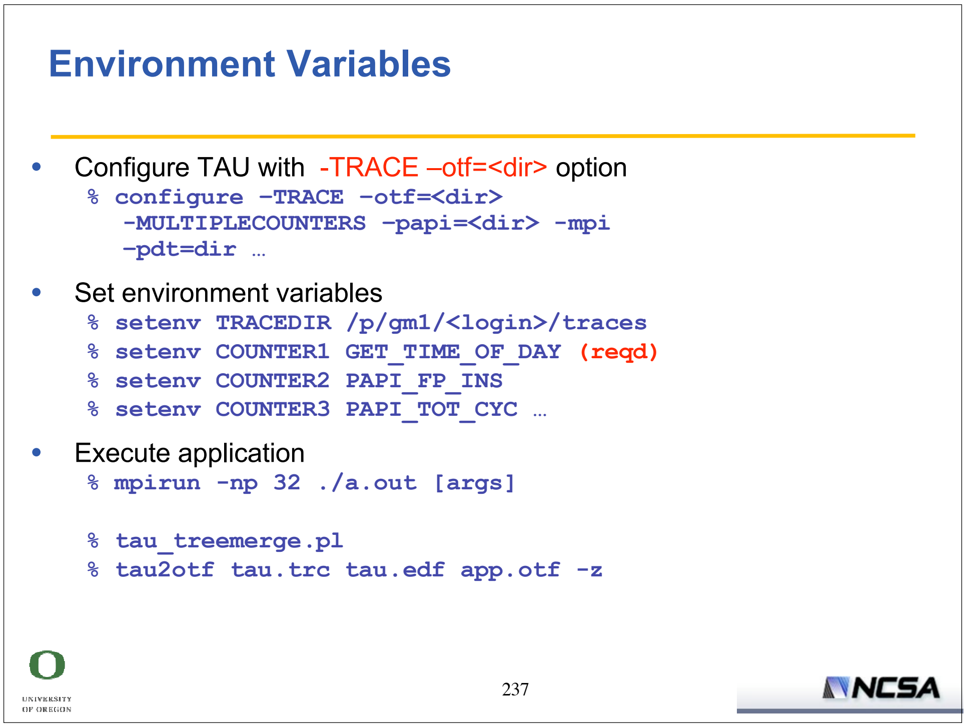
- Instrument and execute application

```
% mpirun -np 4 app
```

- Merge and convert trace files to OTF format

```
% tau_treemerge.pl
% tau2otf tau.trc tau.edf app.otf [-z][-n <nstreams>]
% vampir app.otf
```

OR use VNG to analyze OTF/VTF trace files



Environment Variables

- Configure TAU with -TRACE –otf=<dir> option

```
% configure -TRACE -otf=<dir>
   -MULTIPLECOUNTERS -papi=<dir> -mpi
   -pdt=dir …
```

- Set environment variables

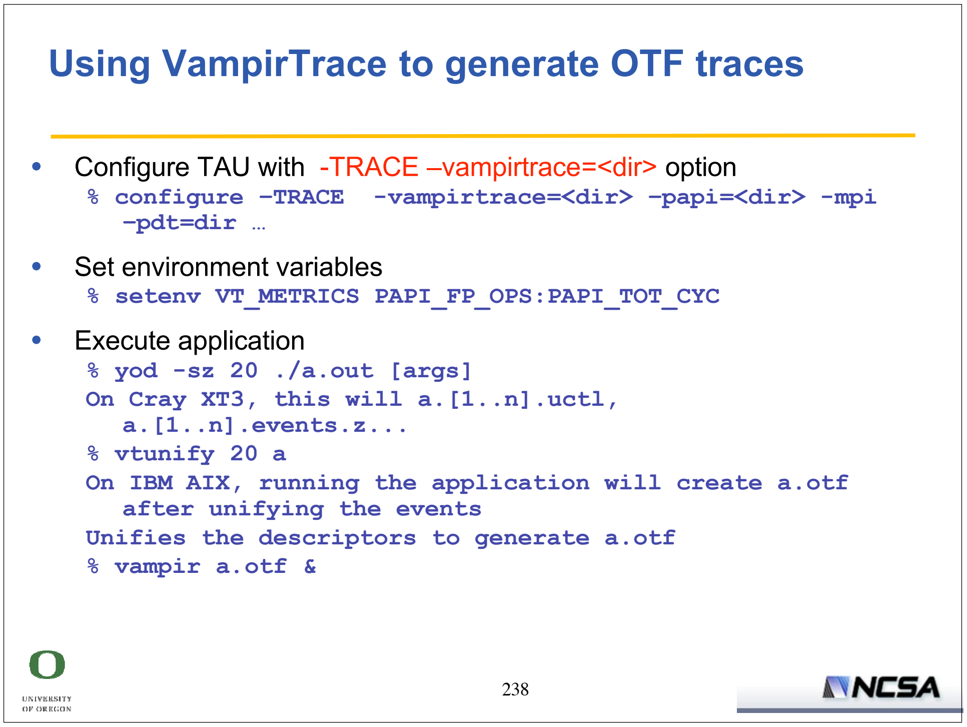
```
% setenv TRACEDIR /p/gm1/<login>/traces
% setenv COUNTER1 GET_TIME_OF_DAY (reqd)
% setenv COUNTER2 PAPI_FP_INS
% setenv COUNTER3 PAPI_TOT_CYC …
```

- Execute application

```
% mpirun -np 32 ./a.out [args]

% tau_treemerge.pl
% tau2otf tau.trc tau.edf app.otf -z
```

UNIVERSITY OF OREGON

NCSA



Using VampirTrace to generate OTF traces

- Configure TAU with -TRACE –vampirtrace=<dir> option

```
% configure -TRACE  -vampirtrace=<dir> -papi=<dir> -mpi
   -pdt=dir …
```

- Set environment variables

```
% setenv VT_METRICS PAPI_FP_OPS:PAPI_TOT_CYC
```

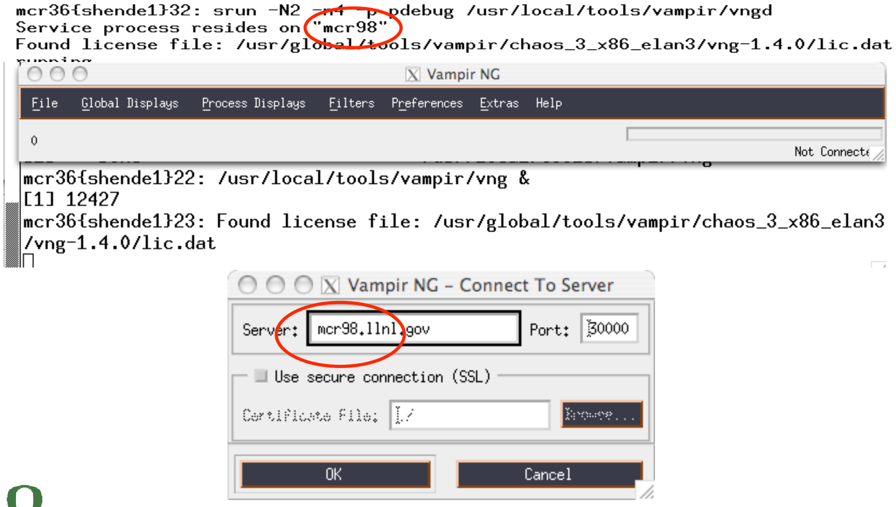
- Execute application

```
% yod -sz 20 ./a.out [args]
On Cray XT3, this will a.[1..n].uctl,
   a.[1..n].events.z...
% vtunify 20 a
On IBM AIX, running the application will create a.otf
   after unifying the events
Unifies the descriptors to generate a.otf
% vampir a.otf &
```

UNIVERSITY OF OREGON

NCSA

Using Vampir Next Generation (VNG v1.4)



```
mcr36{shende1}32: srun -N2 -n4 -p pdebug /usr/local/tools/vampir/vngd
Service process resides on "mcr98"
Found license file: /usr/global/tools/vampir/chaos_3_x86_elan3/vng-1.4.0/lic.dat
```

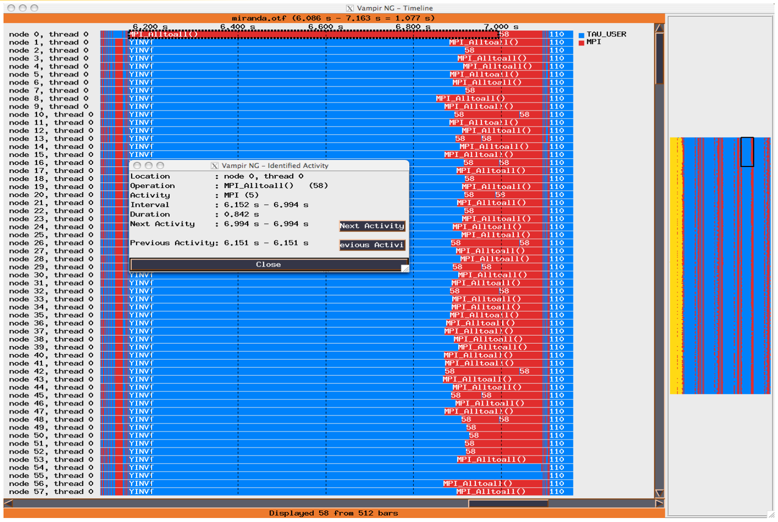
```
mcr36{shende1}22: /usr/local/tools/vampir/vng &
[1] 12427
mcr36{shende1}23: Found license file: /usr/global/tools/vampir/chaos_3_x86_elan3
/vng-1.4.0/lic.dat
```

Server: mcr98.llnl.gov Port: 30000

□ Use secure connection (SSL)

Certificate File: ./

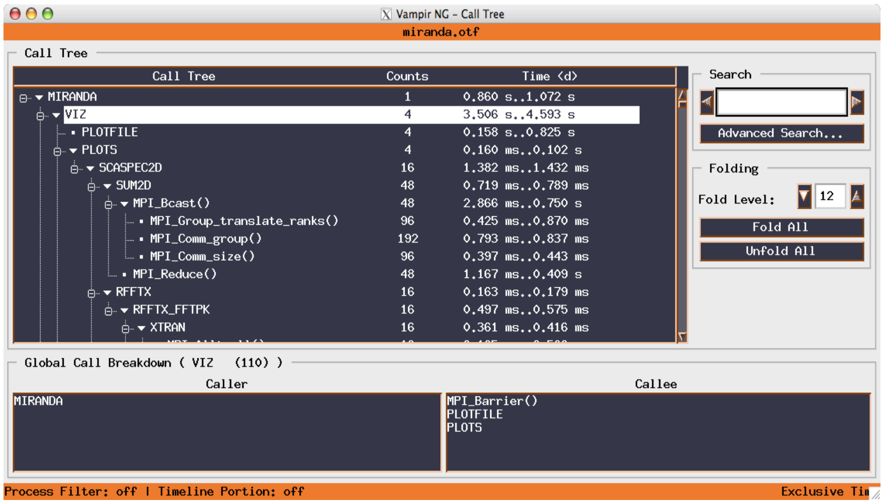
VNG Timeline Display



Vampir NG – Identified Activity

```
Location          : node 0, thread 0
Operation         : MPI_Alltoall()   (58)
Activity          : MPI (5)
Interval          : 6.152 s – 6.994 s
Duration          : 0.842 s
Next Activity     : 6.994 s – 6.994 s

Previous Activity: 6.151 s – 6.151 s
```

VNG Calltree Display



Vampir NG - Call Tree

miranda.otf

Call Tree

Call Tree	Counts	Time (d)
MIRANDA	1	0.860 s..1.072 s
VIZ	4	3.506 s..4.593 s
PLOTFILE	4	0.158 s..0.825 s
PLOTS	4	0.160 ms..0.102 s
SCASPEC2D	16	1.382 ms..1.432 ms
SUM2D	48	0.719 ms..0.789 ms
MPI_Bcast()	48	2.866 ms..0.750 s
MPI_Group_translate_ranks()	96	0.425 ms..0.870 ms
MPI_Comm_group()	192	0.793 ms..0.837 ms
MPI_Comm_size()	96	0.397 ms..0.443 ms
MPI_Reduce()	48	1.167 ms..0.409 s
RFFTX	16	0.163 ms..0.179 ms
RFFTX_FFTPK	16	0.497 ms..0.575 ms
XTRAN	16	0.361 ms..0.416 ms

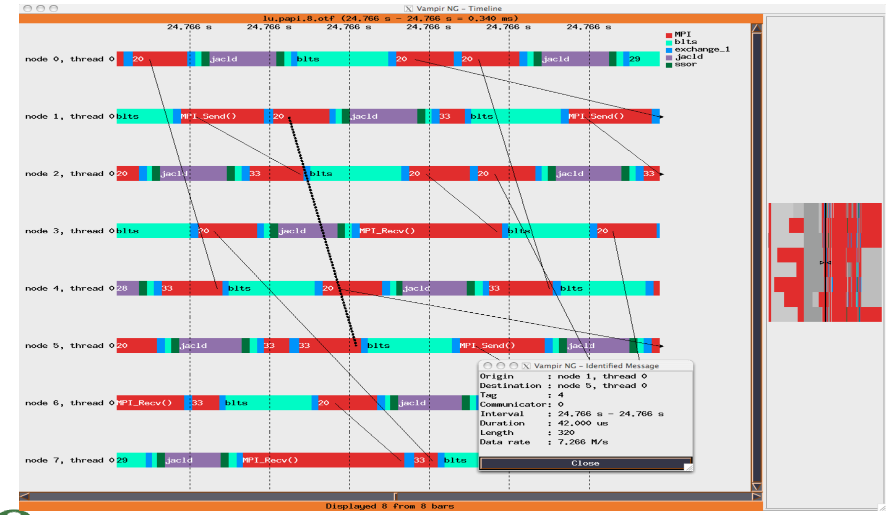
Global Call Breakdown (VIZ (110))

Caller	Callee
MIRANDA	MPI_Barrier() PLOTFILE PLOTS

Process Filter: off | Timeline Portion: off

Exclusive Tim

VNG Timeline Zoomed In

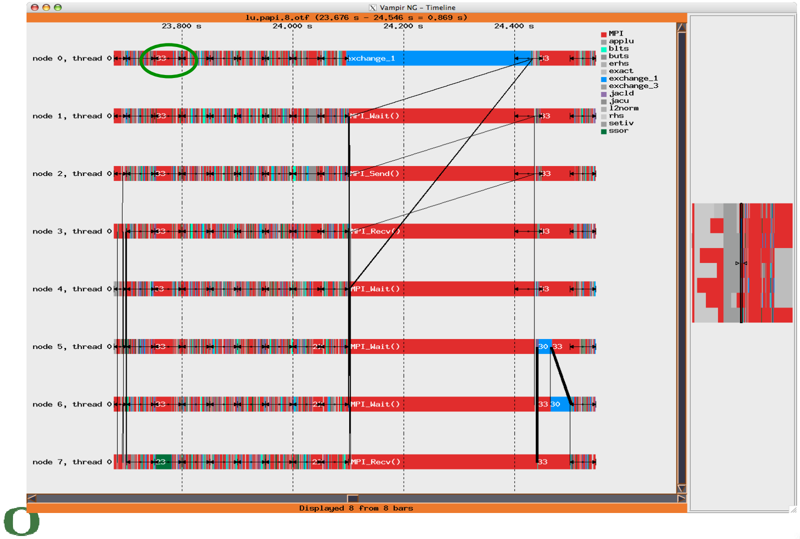


Vampir NG - Identified Message

```
Origin       : node 1, thread 0
Destination  : node 5, thread 0
Tag          : 4
Communicator: 0
Interval     : 24.766 s - 24.766 s
Duration     : 42.000 us
Length       : 320
Data rate    : 7.266 M/s
```

Close

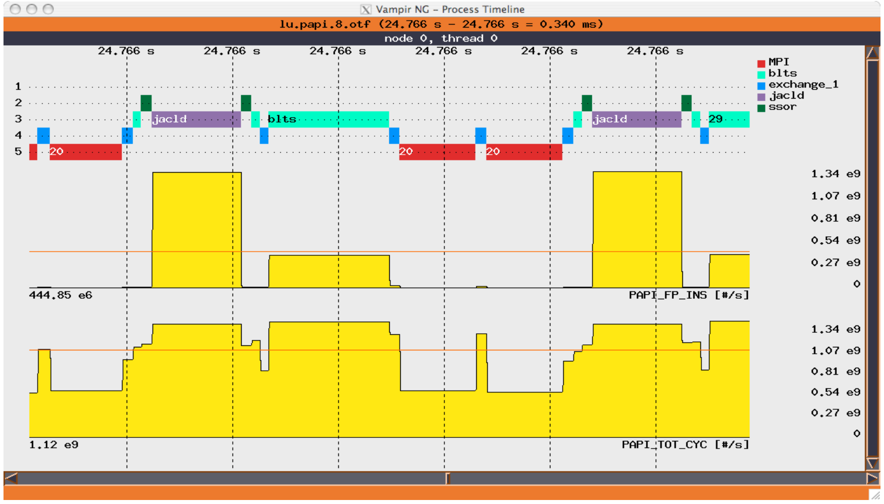
VNG Grouping of Interprocess Communications



UNIVERSITY OF OREGON

NCSA

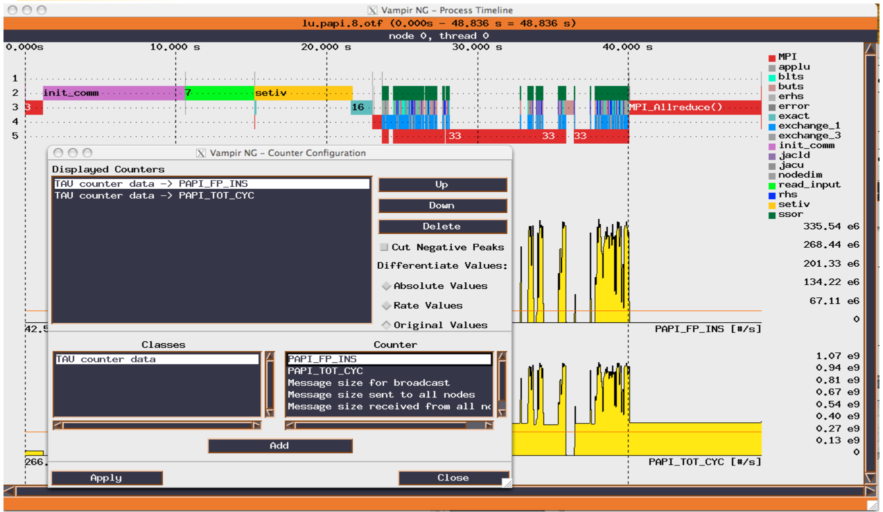
VNG Process Timeline with PAPI Counters



UNIVERSITY OF OREGON

NCSA

OTF/VNG Support for Counters



Displayed Counters

- TAU counter data –> PAPI_FP_INS
- TAU counter data –> PAPI_TOT_CYC

□ Cut Negative Peaks

Differentiate Values:

◇ Absolute Values

◇ Rate Values

◇ Original Values

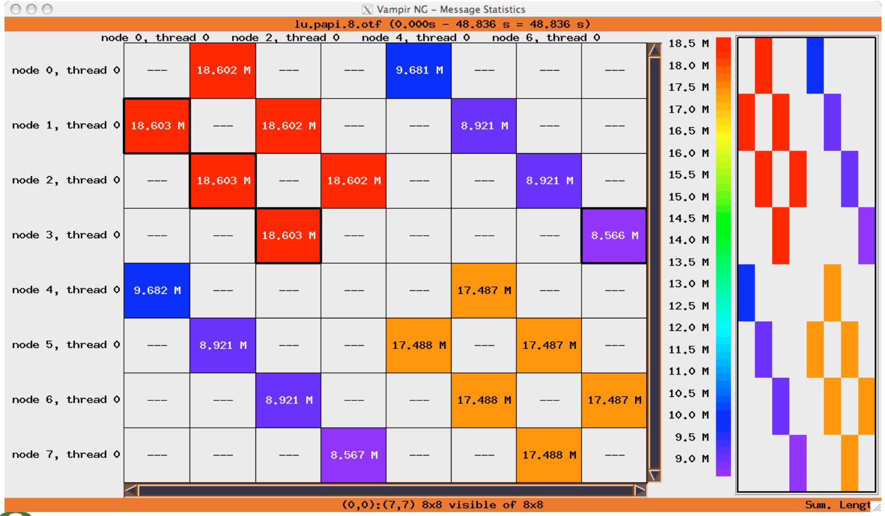
Classes

- TAU counter data

Counter

- PAPI_FP_INS
- PAPI_TOT_CYC
- Message size for broadcast
- Message size sent to all nodes
- Message size received from all no

VNG Communication Matrix Display

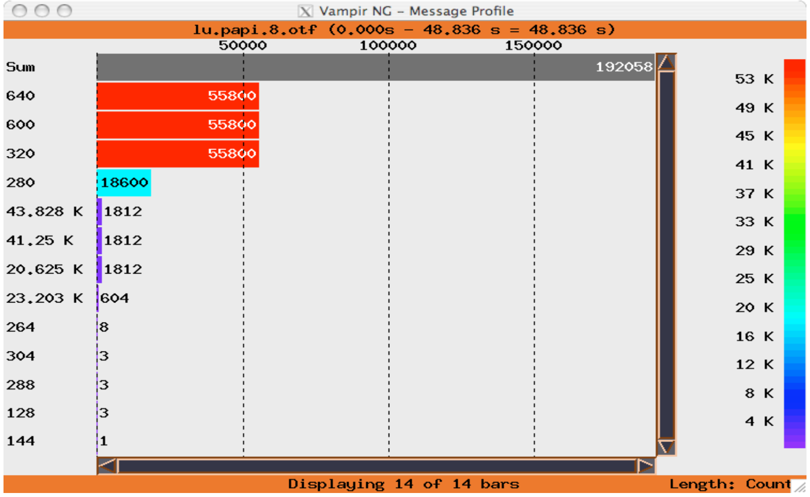


lu.papi.8.otf (0.000s – 48.836 s = 48.836 s)

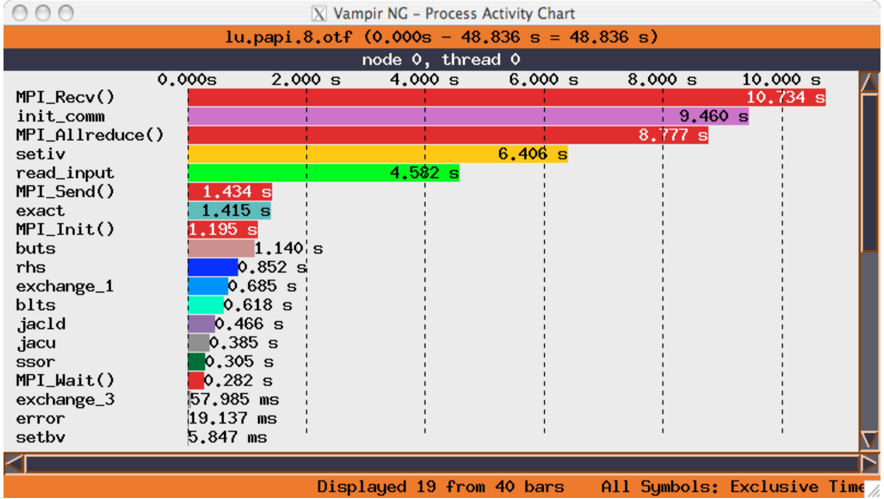
	node 0, thread 0		node 2, thread 0		node 4, thread 0		node 6, thread 0	
node 0, thread 0	---	18.602 M	---	---	9.681 M	---	---	---
node 1, thread 0	18.603 M	---	18.602 M	---	---	8.921 M	---	---
node 2, thread 0	---	18.603 M	---	18.602 M	---	---	8.921 M	---
node 3, thread 0	---	---	18.603 M	---	---	---	---	8.566 M
node 4, thread 0	9.682 M	---	---	---	---	17.487 M	---	---
node 5, thread 0	---	8.921 M	---	---	17.488 M	---	17.487 M	---
node 6, thread 0	---	---	8.921 M	---	---	17.488 M	---	17.487 M
node 7, thread 0	---	---	---	8.567 M	---	---	17.488 M	---

(0,0):(7,7) 8x8 visible of 8x8

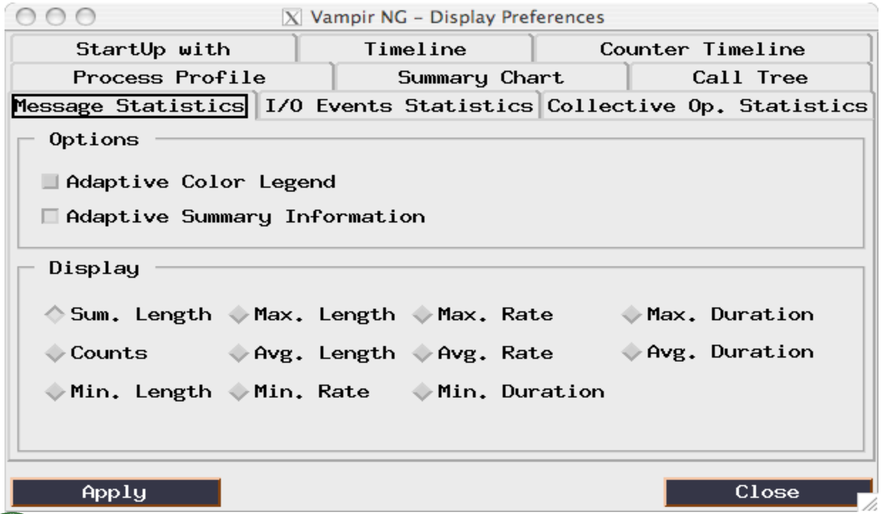
VNG Message Profile



VNG Process Activity Chart



VNG Preferences



Options

□ Adaptive Color Legend

□ Adaptive Summary Information

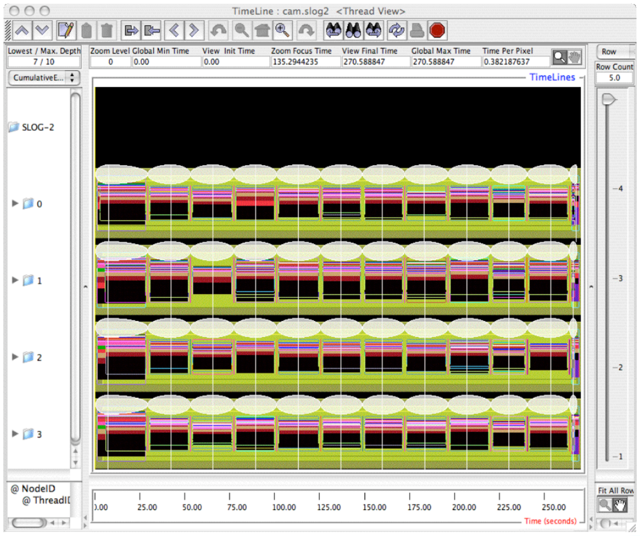
Display

◇ Sum. Length	◇ Max. Length	◇ Max. Rate	◇ Max. Duration
◇ Counts	◇ Avg. Length	◇ Avg. Rate	◇ Avg. Duration
◇ Min. Length	◇ Min. Rate	◇ Min. Duration	

Jumpshot

- http://www-unix.mcs.anl.gov/perfvis/software/viewers/index.htm
- Developed at Argonne National Laboratory as part of the MPICH project
  - Also works with other MPI implementations
  - Jumpshot is bundled with the TAU package
- Java-based tracefile visualization tool for postmortem performance analysis of MPI programs
- Latest version is Jumpshot-4 for SLOG-2 format
  - Scalable level of detail support
  - Timeline and histogram views
  - Scrolling and zooming
  - Search/scan facility

Jumpshot



TimeLine : cam.slog2 <Thread View>

Lowest / Max. Depth: 7 / 10
Zoom Level: 0
Global Min Time: 0.00
View Init Time: 0.00
Zoom Focus Time: 135.2944235
View Final Time: 270.588847
Global Max Time: 270.588847
Time Per Pixel: 0.382187637
Row Count: 5.0

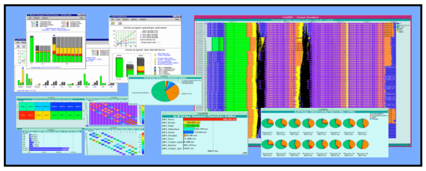
KOJAK Project

- Collaborative research project between
  - Forschungszentrum Jülich
  - University of Tennessee
- Automatic performance analysis
  - MPI and/or OpenMP applications
  - Parallel communication analysis
  - CPU and memory analysis
- WWW
  - http://www.fz-juelich.de/zam/kojak/
  - htttp://icl.cs.utk.edu/kojak/
- Contacts
  - kojak@fz-juelich.de
  - kojak@cs.utk.edu

Enhancing Productivity with Automated Performance Analysis

Efficient development of efficient code

- Tools are needed that help optimize applications by
  - Collecting relevant performance data
  - Automatically identifying the causes of performance problems
- Requirements
  - Expressiveness and accuracy of results
  - Scalability
  - Convenience of use



KOJAK Tools

- KOJAK Trace Analysis Environment
  - Automatic event trace analysis for MPI and / or OpenMP applications
  - Includes tools for trace generation and analysis
    - EPILOG tracing library
    - EXPERT trace analyzer
  - Display of results using CUBE

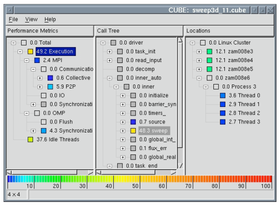
(1) Wolf, Felix: **“Automatic Performance Analysis on Parallel Computers with SMP Nodes”** , Dissertation, RWTH Aachen, NIC Series, Volume 17, Februar 2003. http://www.fz-juelich.de/nic-series/volume17/volume17.html

(2) Wolf F., Mohr, B. **"Automatic performance analysis of hybrid MPI/OpenMP applications,"** *Journal of Systems Architecture, Special Issue 'Evolutions in parallel distributed and network-based processing'*, Clematis, A., D'Agostino, D. eds. Elsevier, 49(10-11), pp. 421-439, November, 2003.

KOJAK Tools (2)

- CUBE
  - Generic display for call-tree profiles
  - Automatic comparison of different experiments
  - Download http://icl.cs.utk.edu/kojak/cube/

Song, F., Wolf, F., Bhatia, N., Dongarra, J., Moore, S. **"An Algebra for Cross-Experiment Performance Analysis,"** *2004 International Conference on Parallel Processing (ICPP-04)*, Montreal, Quebec, Canada, August 2004.



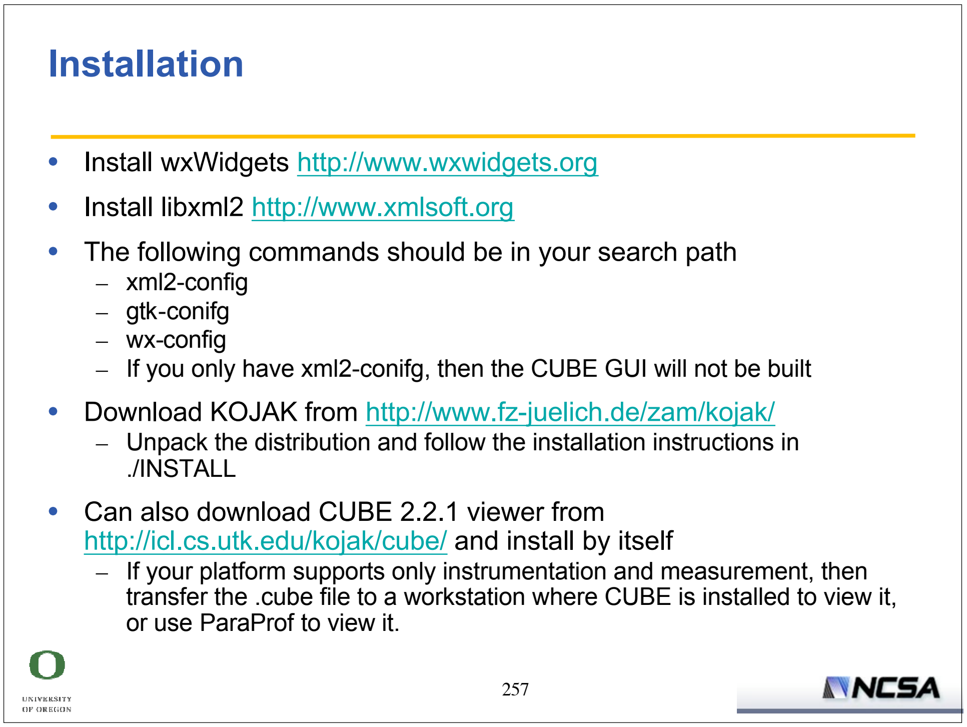
```
0.0 Total
  49.2 Execution
    2.4 MPI
      0.0 Communicatio
        0.6 Collective
        5.9 P2P
      0.0 IO
      0.0 Synchronizati
    0.0 OMP
      0.0 Flush
      4.3 Synchronizati
  37.6 Idle Threads
```

```
0.0 driver
  0.0 task_init
  0.0 read_input
  0.0 decomp
  0.0 inner_auto
    0.0 inner
      0.0 initialize
      0.0 barrier_syn
      0.0 timers_
      0.7 source
      48.3 sweep
      0.0 global_int_
      0.1 flux_err
      0.0 global_real
  0.0 task_end
```

```
0.0 Linux Cluster
  12.1 zam008e3
  12.1 zam008e4
  12.1 zam008e5
  0.0 zam008e6
    0.0 Process 3
      3.6 Thread 0
      2.9 Thread 1
      2.8 Thread 2
      2.7 Thread 3
```

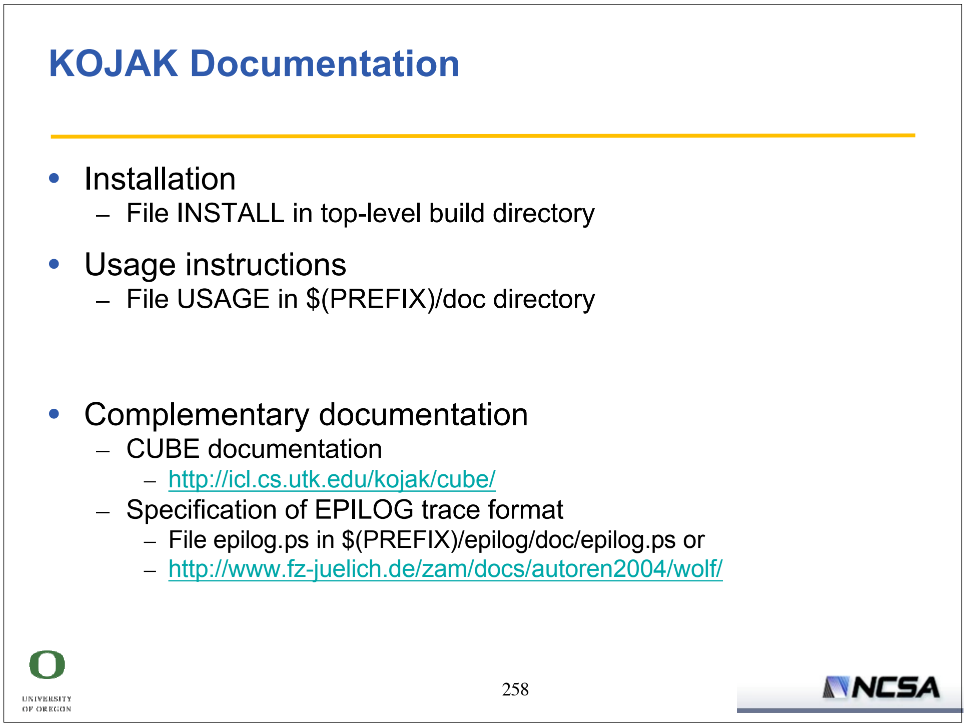
KOJAK: Supported Platforms

- Instrumentation and measurement only (analysis on front-end or workstation)
  - Cray T3E, Cray XD1, Cray X1, and Cray XT3
  - IBM BlueGene/L
  - Hitachi SR-8000
  - NEC SX
- Full support (instrumentation, measurement, and analysis)
  - Linux IA32, IA64, and EMT64/xt3 based clusters
  - IBM AIX Power3/4/5 based clusters
  - SGI Irix MIPS based clusters (Origin 2K, Origin 3K)
  - SGI Linux IA64 based clusters (Altix)
  - SUN Solaris Sparc and x86/xt3 based clusters (SunFire, …)
  - DEC/HP Tru64 Alpha based clusters (Alphaserver, …)



Installation

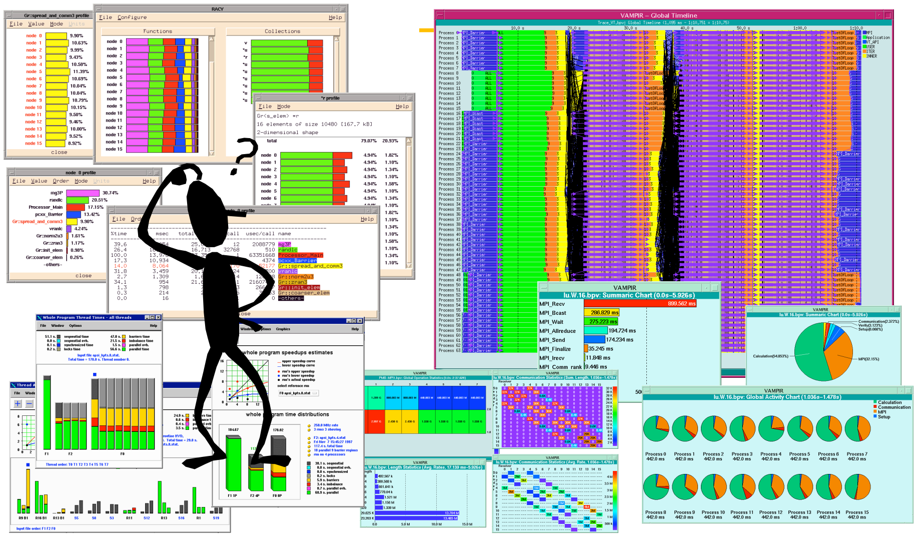
- Install wxWidgets http://www.wxwidgets.org
- Install libxml2 http://www.xmlsoft.org
- The following commands should be in your search path
  - xml2-config
  - gtk-conifg
  - wx-config
  - If you only have xml2-conifg, then the CUBE GUI will not be built
- Download KOJAK from http://www.fz-juelich.de/zam/kojak/
  - Unpack the distribution and follow the installation instructions in ./INSTALL
- Can also download CUBE 2.2.1 viewer from http://icl.cs.utk.edu/kojak/cube/ and install by itself
  - If your platform supports only instrumentation and measurement, then transfer the .cube file to a workstation where CUBE is installed to view it, or use ParaProf to view it.



KOJAK Documentation

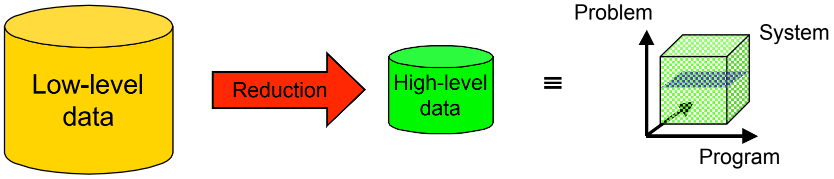
- Installation
  - File INSTALL in top-level build directory
- Usage instructions
  - File USAGE in $(PREFIX)/doc directory
- Complementary documentation
  - CUBE documentation
    - http://icl.cs.utk.edu/kojak/cube/
  - Specification of EPILOG trace format
    - File epilog.ps in $(PREFIX)/epilog/doc/epilog.ps or
    - http://www.fz-juelich.de/zam/docs/autoren2004/wolf/

Low-level View of Performance Behavior

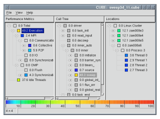


Automatic Performance Analysis

- Transformation of low-level performance data

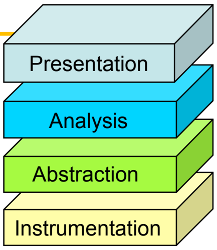


- Take event traces of MPI/OpenMP applications
- Search for execution patterns
- Calculate mapping
  - Problem, call path, system resource ⇒ time
- Display in performance browser



KOJAK Layers

- Instrumentation
  - Insertion of extra code to generate event trace
- Abstract representation of event trace
  - Simplified specification of performance problems
  - Simplified extension of predefined problems
- Automatic analysis
  - Classification and quantification of performance behavior
  - Automatic comparison of multiple experiments
    - Not yet released
- Presentation
  - Navigating / browsing through performance space
  - Can be combined with VAMPIR time-line display

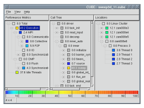


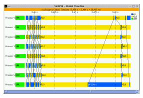
Presentation

Analysis

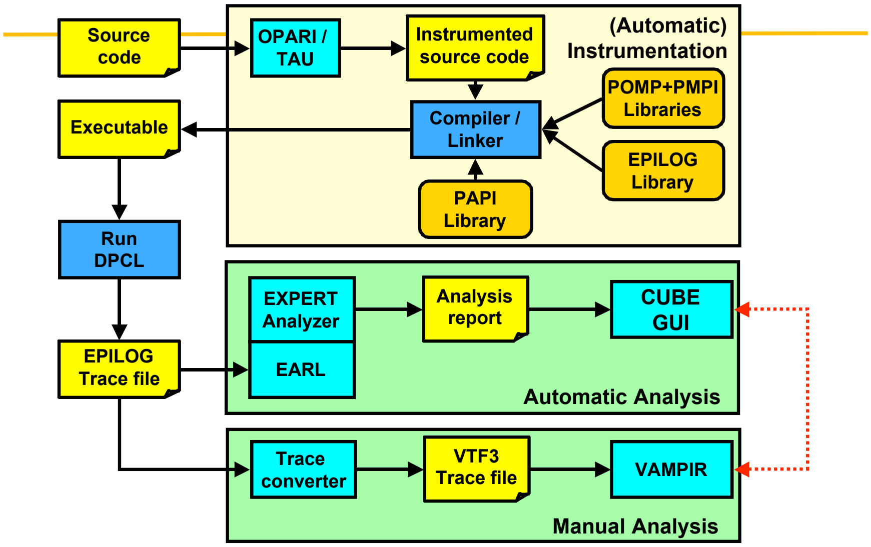
Abstraction

Instrumentation





KOJAK Architecture



Source code

OPARI / TAU

Instrumented source code

(Automatic) Instrumentation

POMP+PMPI Libraries

Executable

Compiler / Linker

EPILOG Library

PAPI Library

Run DPCL

EXPERT Analyzer

Analysis report

CUBE GUI

EPILOG Trace file

EARL

Automatic Analysis

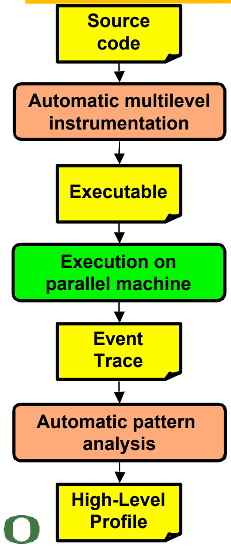
Trace converter

VTF3 Trace file

VAMPIR

Manual Analysis

Analysis process



Source code

Automatic multilevel instrumentation

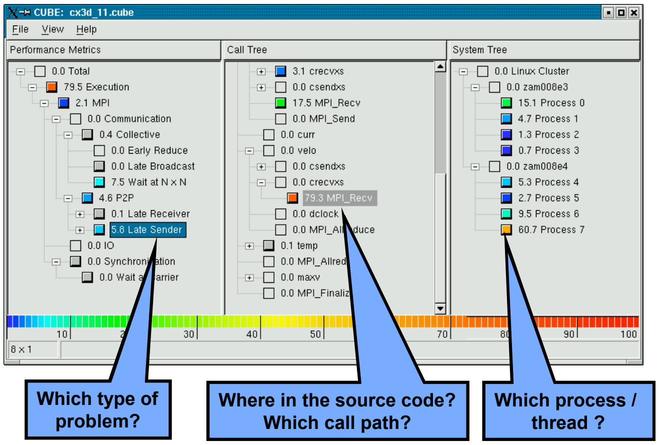
Executable

Execution on parallel machine

Event Trace

Automatic pattern analysis

High-Level Profile



CUBE: cx3d_11.cube

Performance Metrics

```
0.0 Total
  79.5 Execution
    2.1 MPI
      0.0 Communication
        0.4 Collective
          0.0 Early Reduce
          0.0 Late Broadcast
          7.5 Wait at N x N
        4.6 P2P
          0.1 Late Receiver
          5.8 Late Sender
      0.0 IO
      0.0 Synchroni ation
        0.0 Wait a arrier
```

Call Tree

```
    3.1 crecvxs
    0.0 csendxs
    17.5 MPI_Recv
    0.0 MPI_Send
  0.0 curr
  0.0 velo
    0.0 csendxs
    0.0 crecvxs
      79.3 MPI_Recv
    0.0 dclock
    0.0 MPI_All duce
  0.1 temp
  0.0 MPI_Allred
  0.0 maxv
  0.0 MPI_Finaliz
```

System Tree

```
0.0 Linux Cluster
  0.0 zam008e3
    15.1 Process 0
    4.7 Process 1
    1.3 Process 2
    0.7 Process 3
  0.0 zam008e4
    5.3 Process 4
    2.7 Process 5
    9.5 Process 6
    60.7 Process 7
```

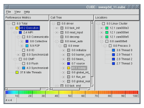
Which type of problem?

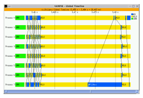
Where in the source code? Which call path?

Which process / thread ?

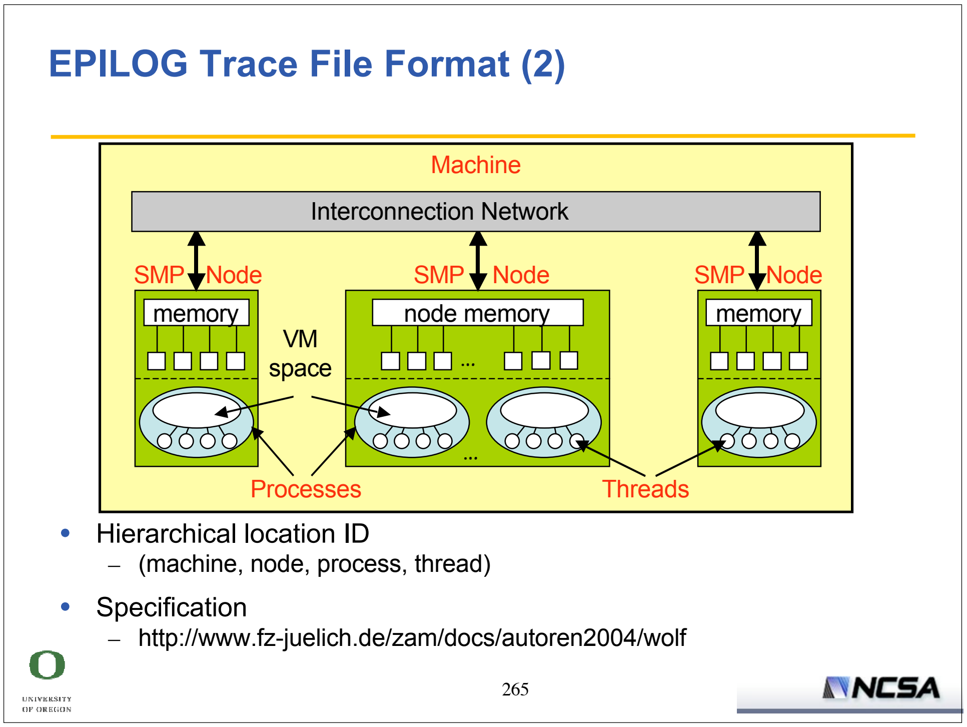
EPILOG Trace File Format

- Event Processing, Investigating, and LOGging
- MPI and OpenMP support (i.e., thread-safe)
  - Region enter and exit
  - Collective region enter and exit (MPI & OpenMP)
  - Message send and receive
  - Parallel region fork and join
  - Lock acquire and release





- Stores source code + HW counter information
- Input of the EXPERT analyzer
- Visualization using VAMPIR
  - EPILOG ⇨VTF3 converter



EPILOG Trace File Format (2)

Machine

Interconnection Network

SMP Node

memory

SMP Node

node memory

SMP Node

memory

VM space

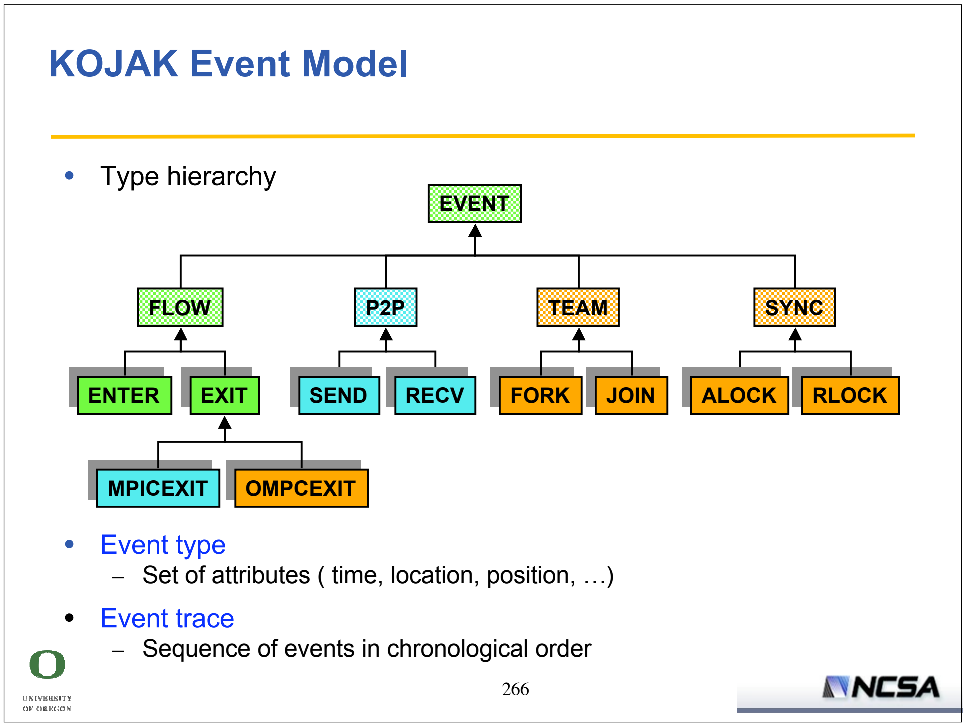
…

…

Processes

Threads

- Hierarchical location ID
  - (machine, node, process, thread)
- Specification
  - http://www.fz-juelich.de/zam/docs/autoren2004/wolf



KOJAK Event Model

- Type hierarchy

EVENT

FLOW

P2P

TEAM

SYNC

ENTER

EXIT

SEND

RECV

FORK

JOIN

ALOCK

RLOCK

MPICEXIT

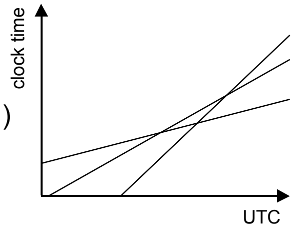
OMPCEXIT

- Event type
  - Set of attributes (time, location, position, …)
- Event trace
  - Sequence of events in chronological order

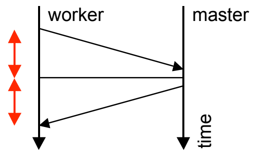
Clock Synchronization

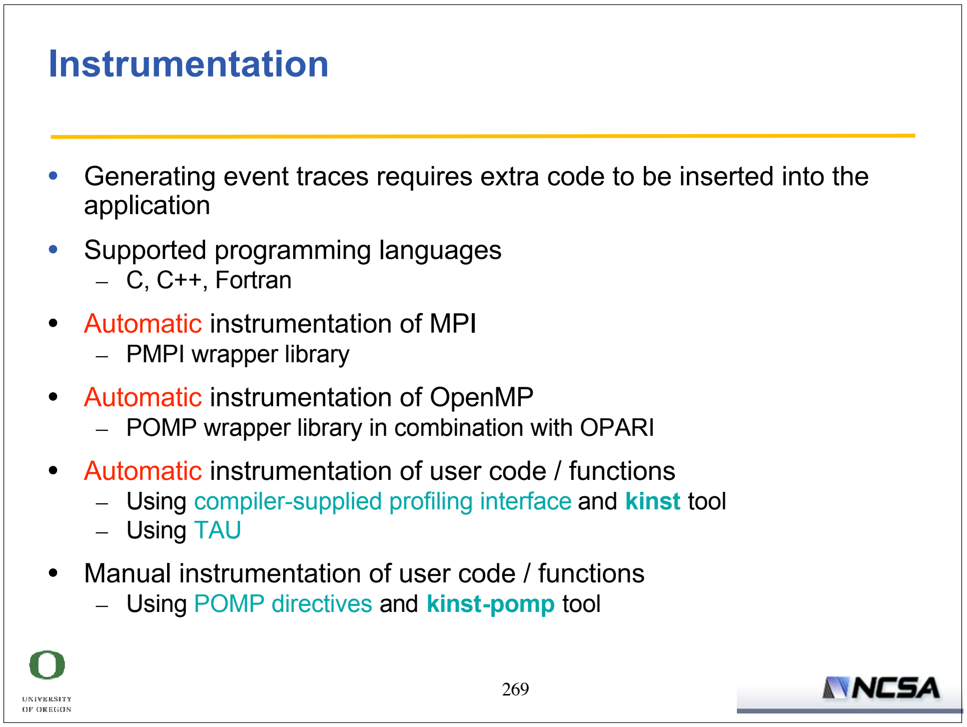
- Time ordering of parallel events requires global time
- Accuracy requirements
  - Correct order of message events (latency!)
- Many clusters provide only distributed local clocks
- Local clocks may differ in drift and offset
  - Drift: clocks may run differently fast
  - Offset: clocks may start at different times
- Clock synchronization
  - Hardware: cannot be changed by tool builder
  - Software: online / offline
- Online: (X)NTP accuracy usually too low

Offline Clock Synchronization



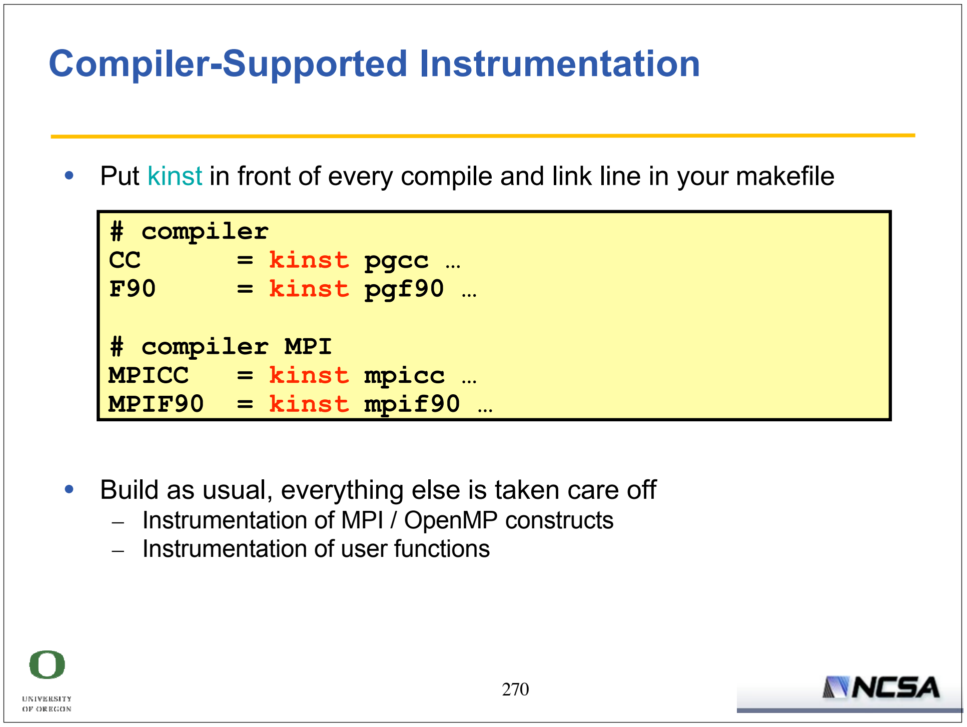
- Model
  - Different offset
  - Different but constant drift (approximation!)
  - One master clock
- Algorithm
  - Measure offset worker ↔ master (2x)
  - Request time from master (Nx)
    - Take shortest propagation time
    - Assume symmetric propagation
  - Get two pairs of (worker time s_i , offset o_i)
  - Master time $m(s) := s + \frac{(o_2 - o_1)}{(s_2 - s_1)} * (s - s_1) + o_1$





Instrumentation

- Generating event traces requires extra code to be inserted into the application
- Supported programming languages
  - C, C++, Fortran
- Automatic instrumentation of MPI
  - PMPI wrapper library
- Automatic instrumentation of OpenMP
  - POMP wrapper library in combination with OPARI
- Automatic instrumentation of user code / functions
  - Using compiler-supplied profiling interface and **kinst** tool
  - Using TAU
- Manual instrumentation of user code / functions
  - Using POMP directives and **kinst-pomp** tool



Compiler-Supported Instrumentation

- Put kinst in front of every compile and link line in your makefile

```
# compiler
CC       = kinst pgcc …
F90      = kinst pgf90 …

# compiler MPI
MPICC    = kinst mpicc …
MPIF90   = kinst mpif90 …
```

- Build as usual, everything else is taken care off
  - Instrumentation of MPI / OpenMP constructs
  - Instrumentation of user functions

Compiler-Supported Instrumentation (2)

- Platforms
  - Linux / PGI
  - HITACHI SR-8000
  - SUN Solaris (Fortran90 only)
  - NEC SX 6

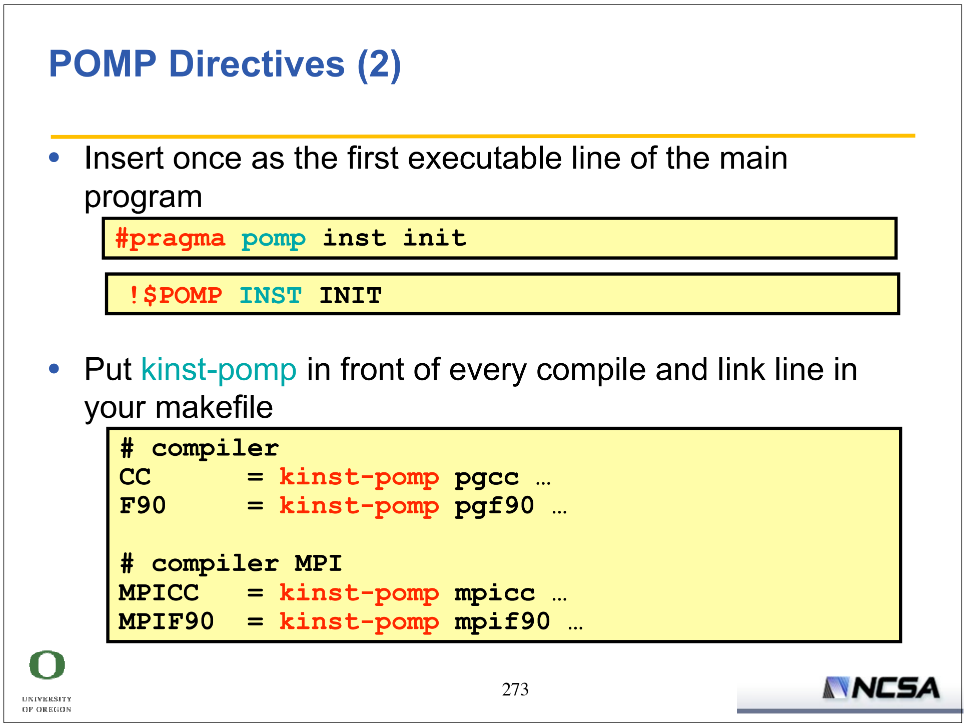
POMP Directives

- Instrumentation of user-specified arbitrary (non-function) code regions
- C/C++

```
#pragma pomp inst begin(name)
     ...
  [ #pragma pomp inst altend(name) ]
     ...
#pragma pomp inst end(name)
```

- Fortran

```
!$POMP INST BEGIN(name)
     ...
  [ !$POMP INST ALTEND(name) ]
     ...
!$POMP INST END(name)
```



POMP Directives (2)

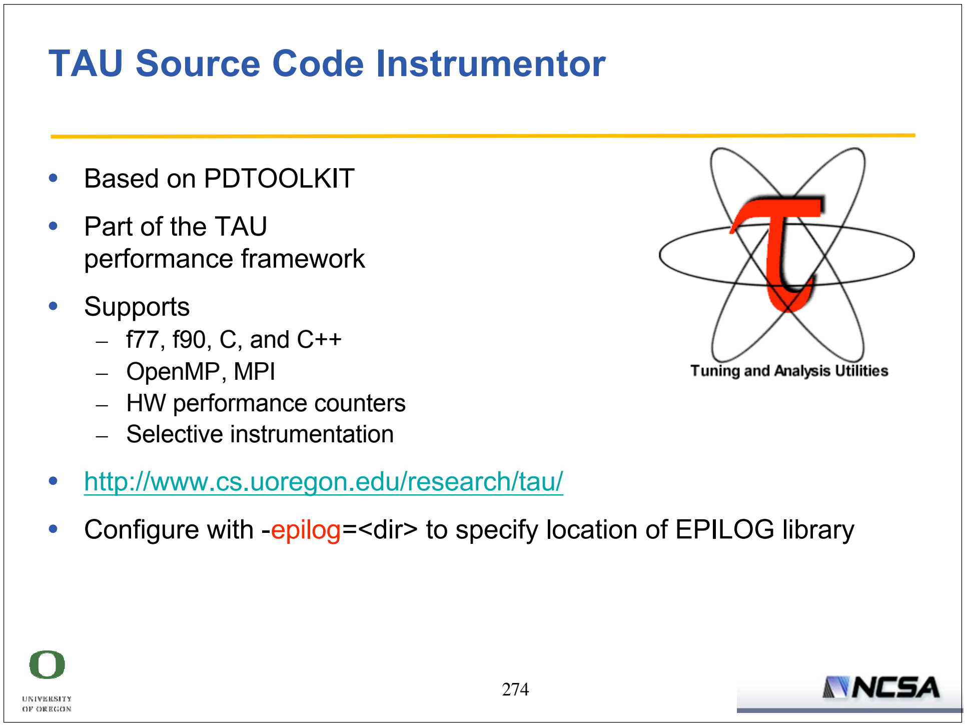
- Insert once as the first executable line of the main program

```
#pragma pomp inst init
```

```
 !$POMP INST INIT
```

- Put kinst-pomp in front of every compile and link line in your makefile

```
# compiler
CC      = kinst-pomp pgcc …
F90     = kinst-pomp pgf90 …

# compiler MPI
MPICC   = kinst-pomp mpicc …
MPIF90  = kinst-pomp mpif90 …
```



TAU Source Code Instrumentor

- Based on PDTOOLKIT
- Part of the TAU performance framework
- Supports
  - f77, f90, C, and C++
  - OpenMP, MPI
  - HW performance counters
  - Selective instrumentation
- http://www.cs.uoregon.edu/research/tau/
- Configure with -epilog=<dir> to specify location of EPILOG library

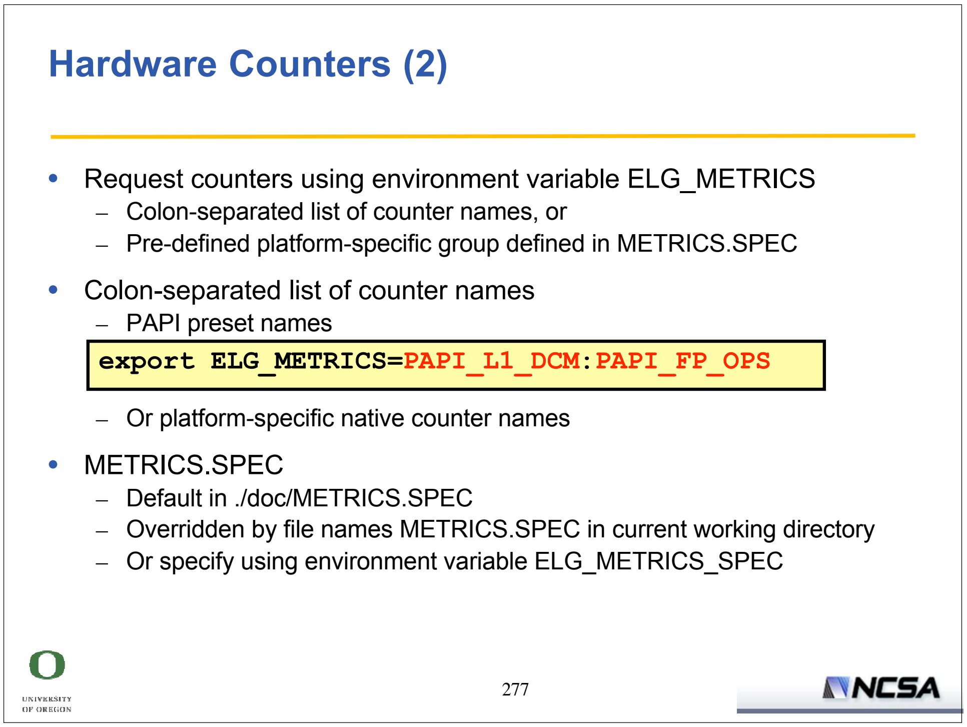
Tuning and Analysis Utilities

KOJAK Runtime Environment

- ELG_PFORM_GDIR
  - Name of global, cluster-wide directory to store final trace file
  - Default platform specific, typically "."
- ELG_PFORM_LDIR
  - Name of node-local directory to store temporary trace files
  - Default platform specific, typically "/tmp"
- ELG_FILE_PREFIX
  - Prefix used for names of EPILOG trace files
  - Default "a"
- ELG_BUFFER_SIZE
  - Size of per-process event trace buffer in bytes
  - Default 10000000
- ELG_VERBOSE
  - Print EPILOG related control information during measurement
  - Default no

Hardware Counters

- Small set of CPU registers that count events
  - Events: signal related to a processor’s function
- Original purpose
  - Verification and evaluation of CPU design
- Can help answer question
  - How efficiently is my application mapped onto the underlying architecture?
- KOJAK supports hardware counter analysis
  - Can be recorded as part of ENTER/EXIT event records
- Uses PAPI for portable access to counters

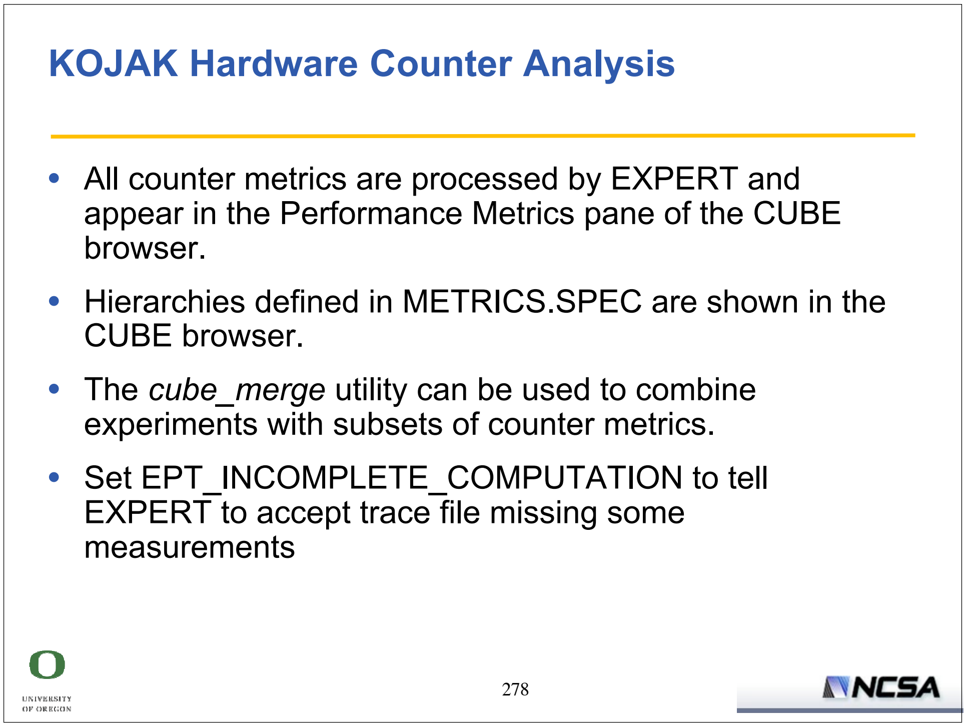


Hardware Counters (2)

- Request counters using environment variable ELG_METRICS
  - Colon-separated list of counter names, or
  - Pre-defined platform-specific group defined in METRICS.SPEC
- Colon-separated list of counter names
  - PAPI preset names

```
export ELG_METRICS=PAPI_L1_DCM:PAPI_FP_OPS
```

  - Or platform-specific native counter names
- METRICS.SPEC
  - Default in ./doc/METRICS.SPEC
  - Overridden by file names METRICS.SPEC in current working directory
  - Or specify using environment variable ELG_METRICS_SPEC



KOJAK Hardware Counter Analysis

- All counter metrics are processed by EXPERT and appear in the Performance Metrics pane of the CUBE browser.
- Hierarchies defined in METRICS.SPEC are shown in the CUBE browser.
- The *cube_merge* utility can be used to combine experiments with subsets of counter metrics.
- Set EPT_INCOMPLETE_COMPUTATION to tell EXPERT to accept trace file missing some measurements

Running the Application

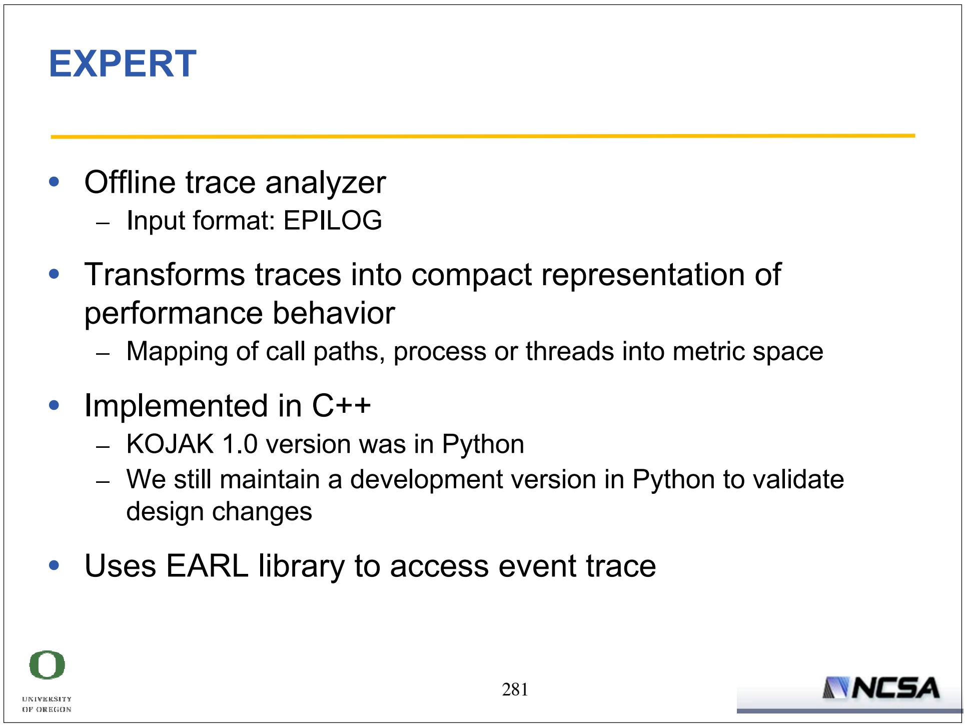
- Run your instrumented application

-
  ```
  > mpirun -np 4 a.out
  > ls
  a.elg  a.out  …
  ```

- KOJAK includes several tools to check trace file
  - ASCII representation of trace file
    - elg_print <file>
  - Event counts and simple statisitics
    - elg_stat <file>
  - Correct order of SEND and RECV events
    - elg_msgord <file>
    - Needed to check correct clock synchronization

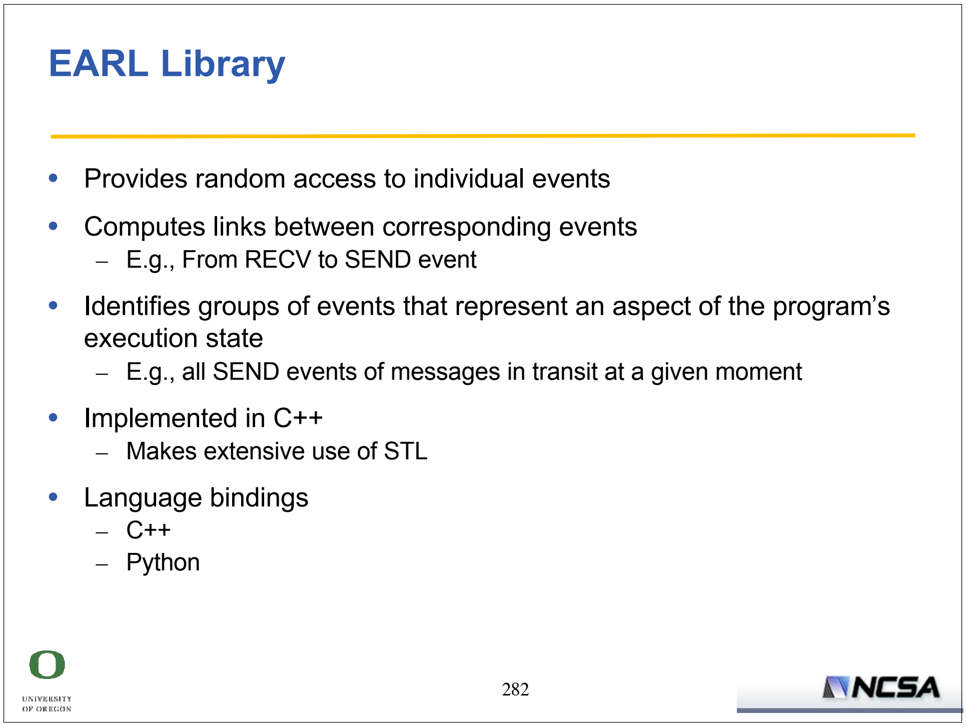
Simple Statistics with elg_stat

```
        ENTER :   119    90   119    90
         EXIT :    71    54    71    54
     MPI_SEND :     0     0     0     0
     MPI_RECV :     0     0     0     0
 MPI_COLLEXIT :    12     0    12     0
     OMP_FORK :     9     0     9     0
     OMP_JOIN :     9     0     9     0
    OMP_ALOCK :     0     0     0     0
    OMP_RLOCK :     0     0     0     0
 OMP_COLLEXIT :    36    36    36    36
     ENTER_CS :     0     0     0     0

                MPI_Barrier :    18 :     9     0     9     0
                  MPI_Bcast :     6 :     3     0     3     0
              MPI_Comm_free :     2 :     1     0     1     0
             MPI_Comm_split :     2 :     1     0     1     0
               MPI_Finalize :     2 :     1     0     1     0
                   MPI_Init :     2 :     1     0     1     0
                       step :   216 :    54    54    54    54
                 sequential :    18 :     9     0     9     0
             !$omp parallel :    36 :     9     9     9     9
             !$omp ibarrier :    36 :     9     9     9     9
                  !$omp for :    36 :     9     9     9     9
             !$omp ibarrier :    36 :     9     9     9     9
                   parallel :     6 :     3     0     3     0
                       main :     2 :     1     0     1     0
```



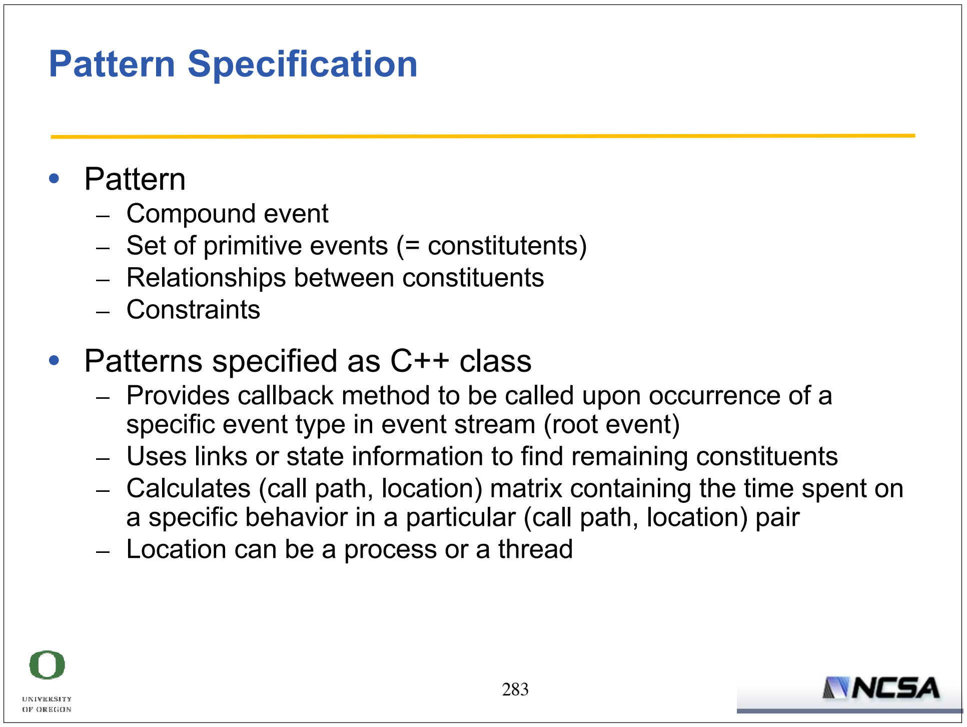
EXPERT

- Offline trace analyzer
  - Input format: EPILOG
- Transforms traces into compact representation of performance behavior
  - Mapping of call paths, process or threads into metric space
- Implemented in C++
  - KOJAK 1.0 version was in Python
  - We still maintain a development version in Python to validate design changes
- Uses EARL library to access event trace



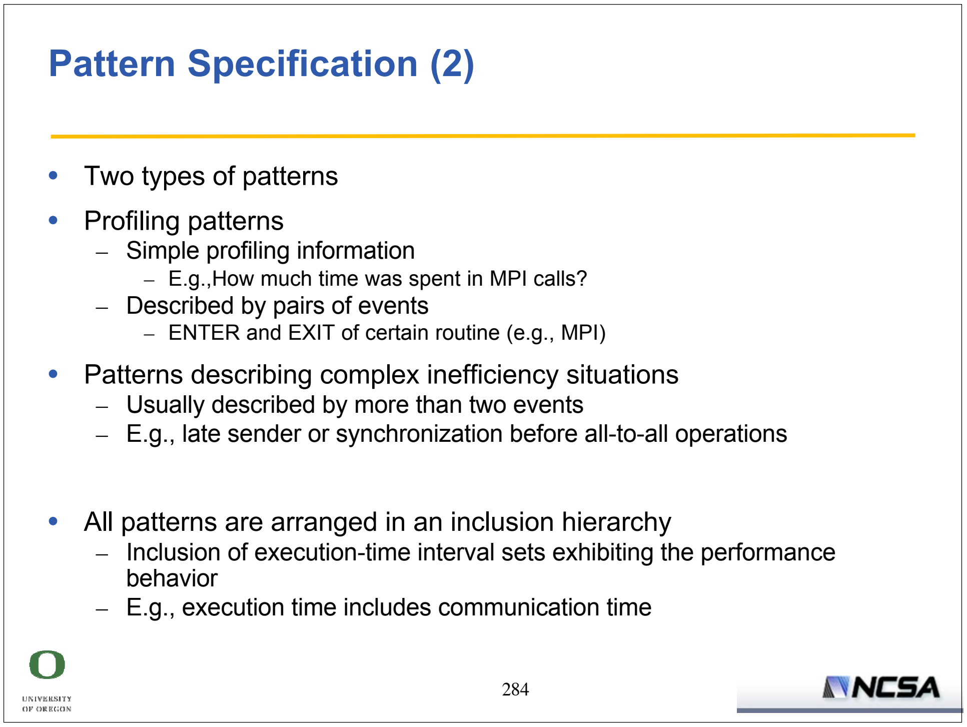
EARL Library

- Provides random access to individual events
- Computes links between corresponding events
  - E.g., From RECV to SEND event
- Identifies groups of events that represent an aspect of the program's execution state
  - E.g., all SEND events of messages in transit at a given moment
- Implemented in C++
  - Makes extensive use of STL
- Language bindings
  - C++
  - Python



Pattern Specification

- Pattern
  - Compound event
  - Set of primitive events (= constitutents)
  - Relationships between constituents
  - Constraints
- Patterns specified as C++ class
  - Provides callback method to be called upon occurrence of a specific event type in event stream (root event)
  - Uses links or state information to find remaining constituents
  - Calculates (call path, location) matrix containing the time spent on a specific behavior in a particular (call path, location) pair
  - Location can be a process or a thread

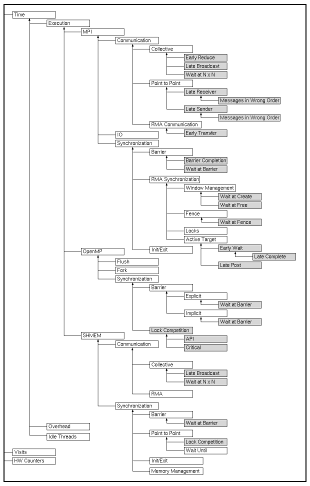


Pattern Specification (2)

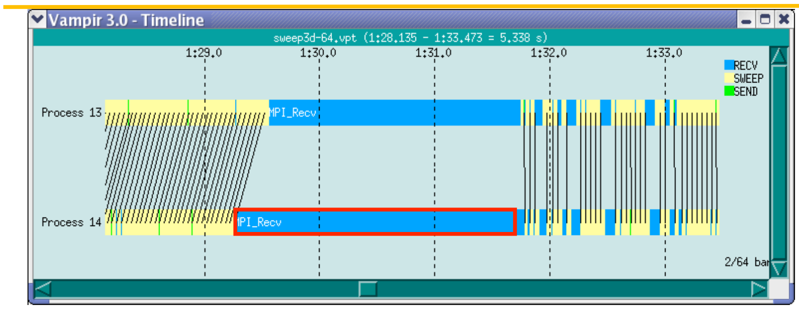
- Two types of patterns
- Profiling patterns
  - Simple profiling information
    - E.g.,How much time was spent in MPI calls?
  - Described by pairs of events
    - ENTER and EXIT of certain routine (e.g., MPI)
- Patterns describing complex inefficiency situations
  - Usually described by more than two events
  - E.g., late sender or synchronization before all-to-all operations
- All patterns are arranged in an inclusion hierarchy
  - Inclusion of execution-time interval sets exhibiting the performance behavior
  - E.g., execution time includes communication time

Pattern hierarchy

- MPI
  - Point-to-point communication
  - Collective communication
  - Barrier synchronization
  - RMA synchronization
- OpenMP
  - Lock synchronization
  - Critical section synchronization
  - Barrier synchronization
- SHMEM
  - Collective communication
  - Barrier / lock synchronization

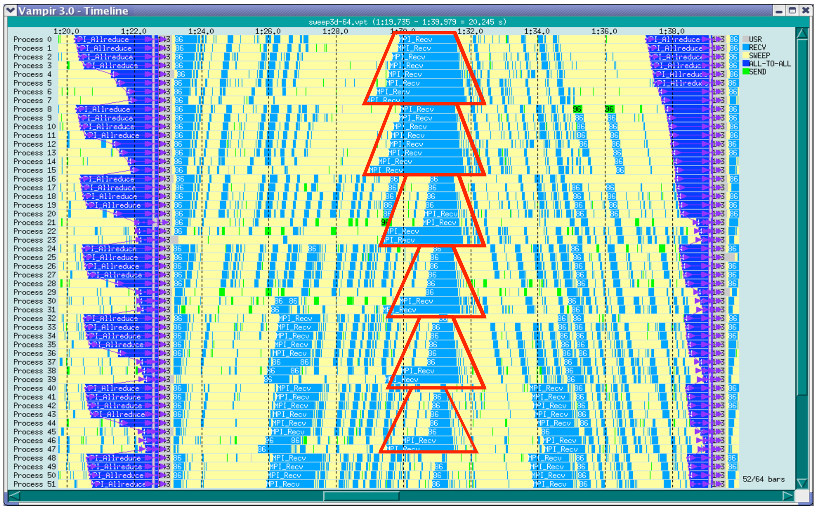


Micro-patterns

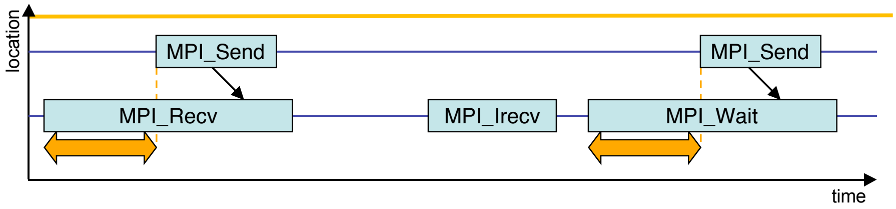


- Undesired wait states as a result of untimely arrival of processes or threads at synchronization points
  - Late sender
  - Wait at n-to-n

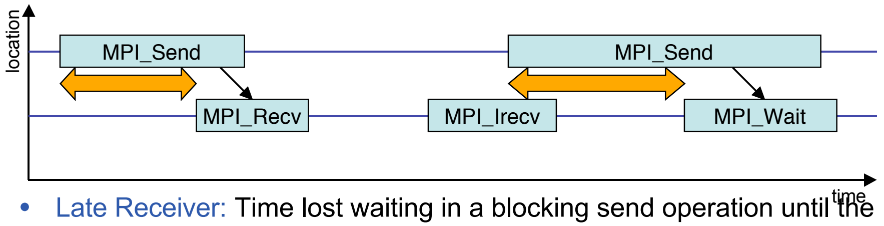
Macro-patterns



KOJAK MPI-1 Pattern: Late Sender / Receiver

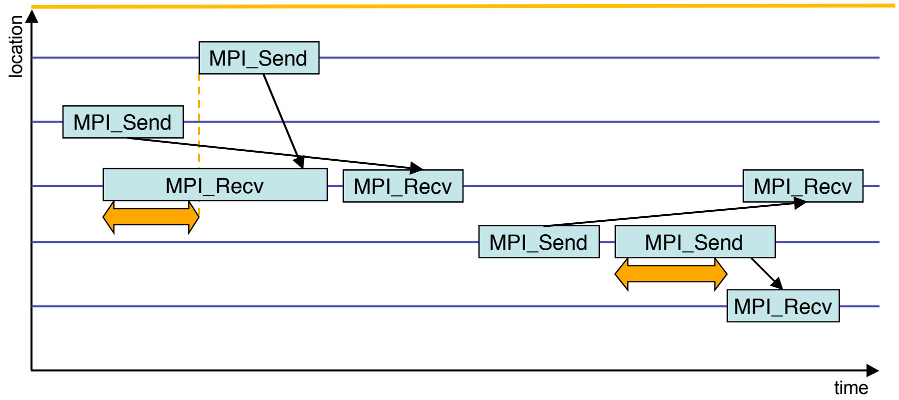


- Late Sender: Time lost waiting caused by a blocking receive operation posted earlier than the corresponding send operation



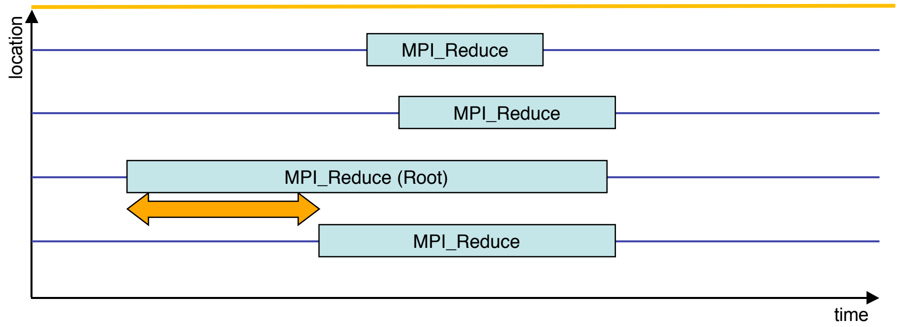
- Late Receiver: Time lost waiting in a blocking send operation until the corresponding receive operation is called

KOJAK MPI-1 Pattern: Wrong Order



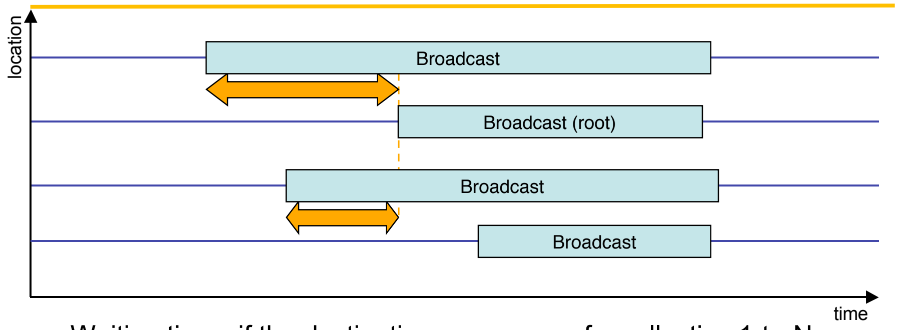
- Late Sender / Receiver patterns caused by messages received/sent in wrong order
- Sub patterns of Late Sender / Receiver

KOJAK MPI-1 Collective Pattern: Early Reduce



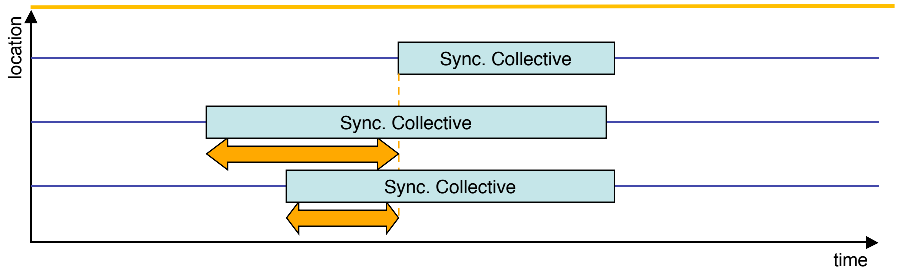
- Waiting time if the destination process (root) of a collective N-to-1 communication operation enters the operation earlier than its sending counterparts
- Applies to MPI calls MPI_Reduce(), MPI_Gather(), MPI_Gatherv()

KOJAK Collective Pattern: Late Broadcast

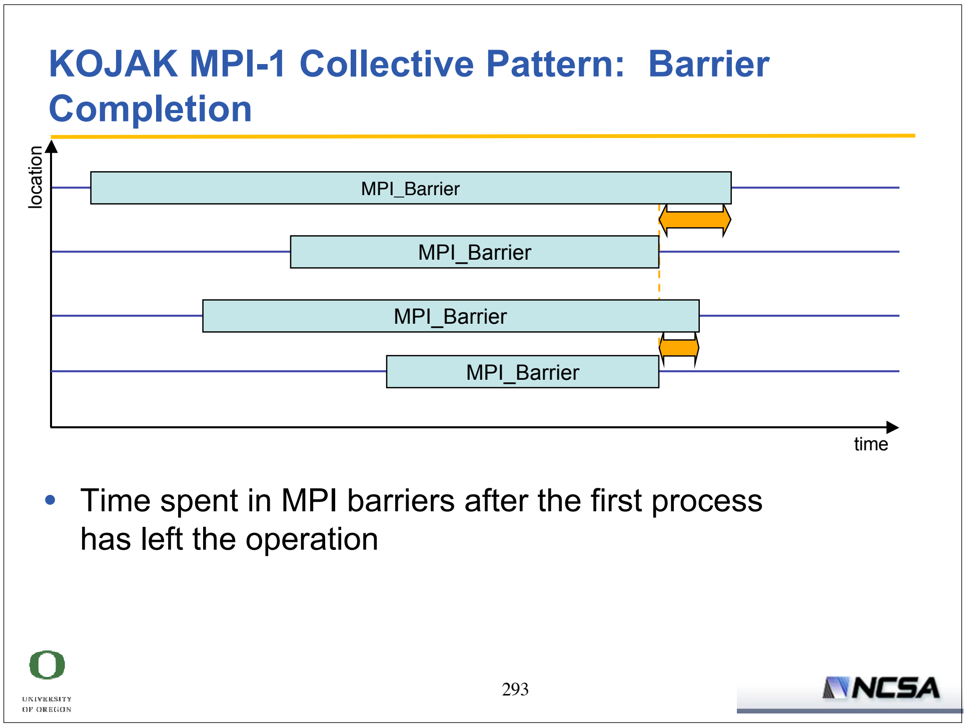


- Waiting times if the destination processes of a collective 1-to-N communication operation enter the operation earlier than the source process (root)
- MPI-1: Applies to MPI_Bcast(), MPI_Scatter(), MPI_Scatterv()
- SHMEM: Applies to shmem_broadcast()

KOJAK Generic Pattern: Wait at

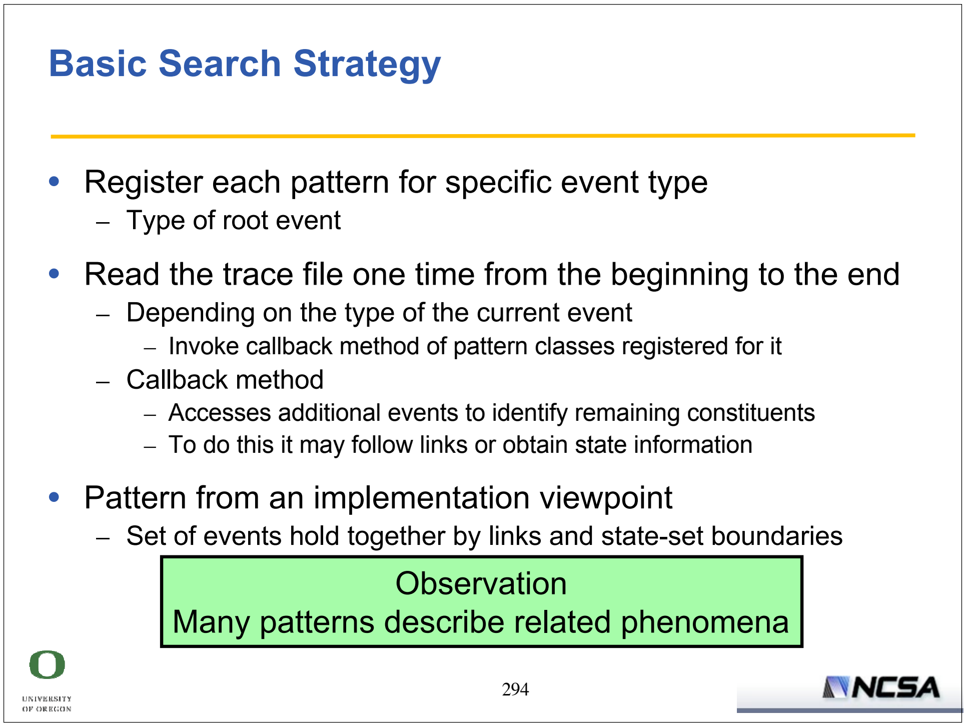


- Time spent waiting in front of a collective synchronizing operation call until the last process reaches the operation
- Pattern instances:
  - Wait at NxN (MPI)
  - Wait at Barrier (MPI)
  - Wait at NxN (SHMEM)
  - Wait at Barrier (SHMEM)
  - Wait at Barrier (OpenMP)
  - Wait at Create (MPI-2)
  - Wait at Free (MPI-2)
  - Wait at Fence (MPI-2)



KOJAK MPI-1 Collective Pattern: Barrier Completion

- Time spent in MPI barriers after the first process has left the operation



Basic Search Strategy

- Register each pattern for specific event type
  - Type of root event
- Read the trace file one time from the beginning to the end
  - Depending on the type of the current event
    - Invoke callback method of pattern classes registered for it
  - Callback method
    - Accesses additional events to identify remaining constituents
    - To do this it may follow links or obtain state information
- Pattern from an implementation viewpoint
  - Set of events hold together by links and state-set boundaries

Observation
Many patterns describe related phenomena

Running EXPERT is simple…

- Run your instrumented application. On XT3, it will produce epik_a/

```
> elg_merge <np> epik_a
```

- Application will generate a trace file a.elg
- Run analyzer with trace file as input
- Generate CUBE intput file a.cube

```
> expert a.elg
Total number of events: 11063530
100 %
Elapsed time (h:m:s): 0 : 3 : 34
Events per second: 51698
```

- Invoke CUBE or TAU’s paraprof

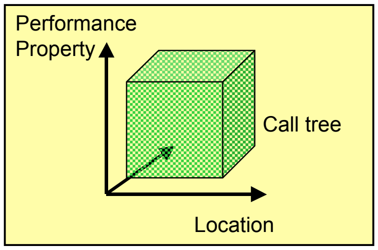
```
> cube a.cube&
> paraprof a.cube &
```

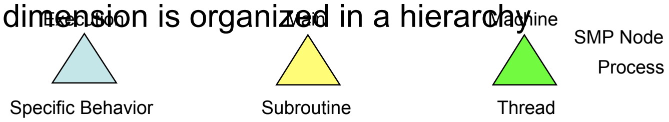
EXPERT Runtime Environment

- EXPERT uses EARL for random-access to individual events during analysis
- Random access based on two different buffer mechanisms
  - Bookmark buffer
    - Stores execution-state information at fixed intervals needed to read the event following the bookmark
  - History buffer
    - Stores contiguous section of the event trace (usually) preceding the most recently accessed event
- Two parameters
  - EARL_BOOKMARK_DISTANCE (default 10000)
  - EARL_HISTORY_SIZE (default 1000 * number of locations)
- Tradeoff between analysis speed and memory consumption

Representation of Performance Behavior

- Three-dimensional matrix
  - Performance property (pattern)
  - Call tree
  - Process or thread
- Uniform mapping onto time
  - Each cell contains fraction of execution time (severity)
  - E.g. waiting time, overhead
- Each dimension is organized in a hierarchy





Parallel vs. Single-Node performance

“The single most important impediment to good parallel performance is *still* poor single-node performance.”

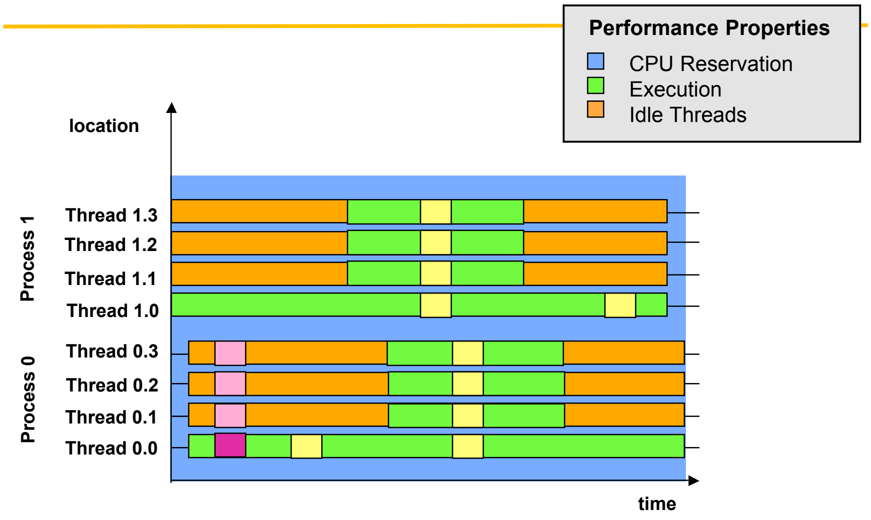
William Gropp

- Increasing performance gap between CPU and memory
  - Annual CPU performance improvement is 55 %
  - Annual memory latency improvement is 7%
- Internal operation of a microprocessor becomes more and more complex
  - Pipelining
  - Out-of-order instruction issuing
  - Branch prediction
  - Non-blocking caches

Single-Node Performance in EXPERT

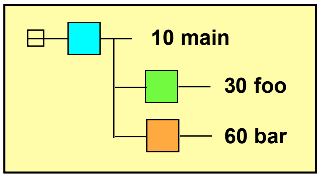
- How do my processes and threads perform individually?
  - CPU performance
  - Memory performance
- Analysis of parallelism performance
  - Temporal and spatial relationships between run-time events
- Analysis of CPU and memory performance
  - Hardware counters
- Analysis
  - EXPERT Identifies tuples (call path, thread) whose occurrence rate of a certain event is above / below a certain threshold
  - Use entire execution time of those tuples as severity (upper bound)

KOJAK Time Model



CUBE GUI

- Design emphasizes simplicity by combining a small number of orthogonal features
- Three coupled tree browsers
- Each node labeled with metric value
- Limited set of actions
- Selecting a metric / call path
  - Break down of aggregated values
- Expanding / collapsing nodes
  - Collapsed node represents entire subtree
  - Expanded node represents only itself without children
- Scalable because level of detail can be adjusted
- Separate documentation: http://icl.cs.utk.edu/kojak/cube/

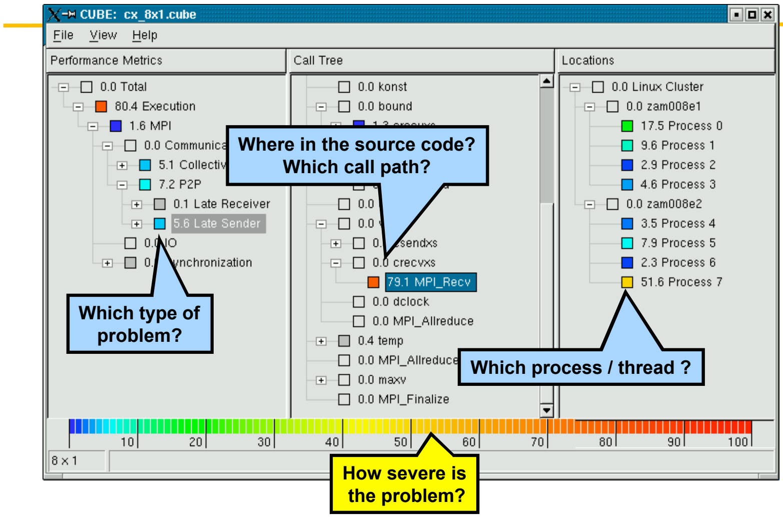


10 main

30 foo

60 bar

CUBE GUI (2)



Performance Metrics:

```
□ 0.0 Total
  ■ 80.4 Execution
    ■ 1.6 MPI
      □ 0.0 Communica
        ■ 5.1 Collectiv
        ■ 7.2 P2P
          ■ 0.1 Late Receiver
          ■ 5.6 Late Sender
      □ 0.0 IO
      ■ 0. ynchronization
```

Call Tree:

```
□ 0.0 konst
□ 0.0 bound
□ 0.0
□ 0.0
  □ 0. sendxs
  □ 0.0 crecvxs
    ■ 79.1 MPI_Recv
  □ 0.0 dclock
  □ 0.0 MPI_Allreduce
■ 0.4 temp
□ 0.0 MPI_Allreduce
□ 0.0 maxv
□ 0.0 MPI_Finalize
```

Locations:

```
□ 0.0 Linux Cluster
  □ 0.0 zam008e1
    ■ 17.5 Process 0
    ■ 9.6 Process 1
    ■ 2.9 Process 2
    ■ 4.6 Process 3
  □ 0.0 zam008e2
    ■ 3.5 Process 4
    ■ 7.9 Process 5
    ■ 2.3 Process 6
    ■ 51.6 Process 7
```

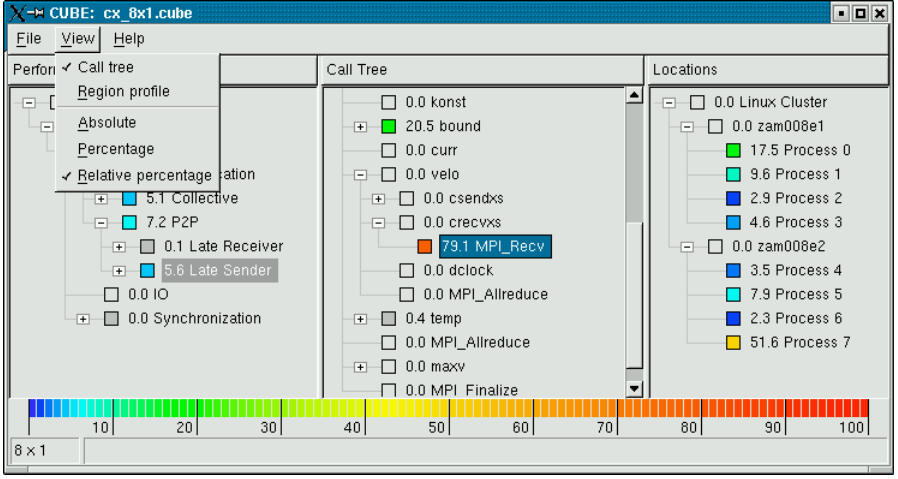
Where in the source code? Which call path?

Which type of problem?

Which process / thread ?

How severe is the problem?

View Options

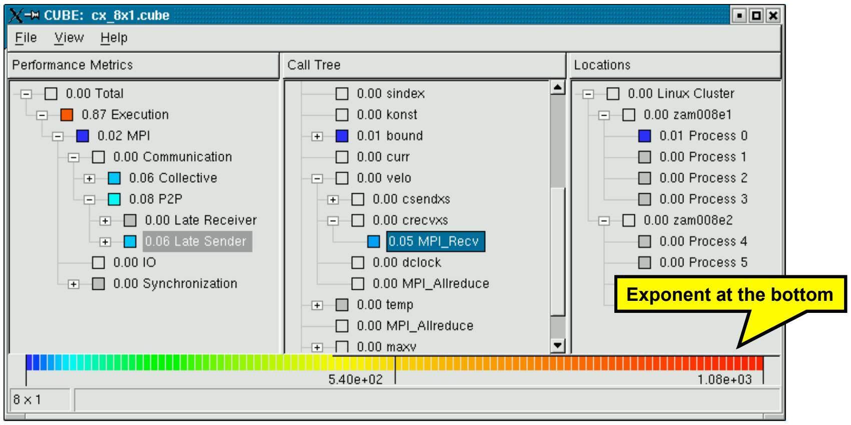


View Options (2)

- Number representation
  - Absolute
    - All values accumulated time values in seconds
    - Scientific notation, color legend shows exponent
  - Percentage
    - All values percentages of the total execution time
  - Relative percentage
    - All values percentages of the selection in the left neighbor tree
- Program resources
  - Call tree
  - Flat region profile
    - Module, region, subregions
- Note that the more general CUBE model also allows for other metrics (e.g., cache misses)

Absolute Mode

- All values accumulated time values in seconds
- Scientific notation, color legend shows exponent



CUBE: cx_8x1.cube

Performance Metrics

- 0.00 Total
  - 0.87 Execution
    - 0.02 MPI
      - 0.00 Communication
        - 0.06 Collective
        - 0.08 P2P
          - 0.00 Late Receiver
          - 0.06 Late Sender
      - 0.00 IO
      - 0.00 Synchronization

Call Tree

- 0.00 sindex
- 0.00 konst
- 0.01 bound
- 0.00 curr
- 0.00 velo
  - 0.00 csendxs
  - 0.00 crecvxs
    - 0.05 MPI_Recv
  - 0.00 dclock
  - 0.00 MPI_Allreduce
- 0.00 temp
- 0.00 MPI_Allreduce
- 0.00 maxv

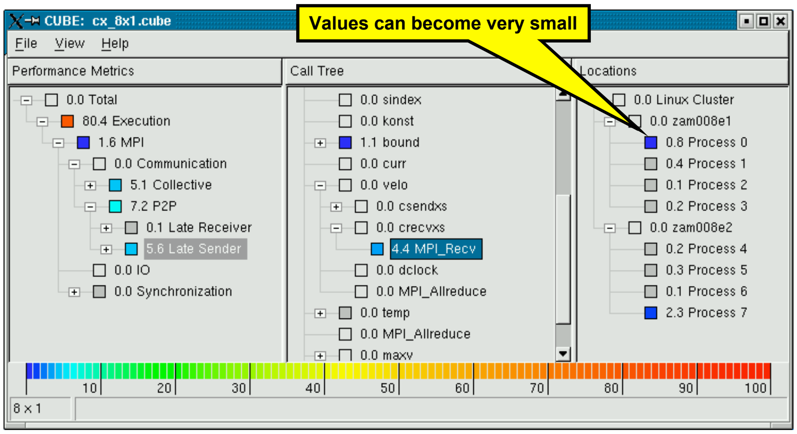
Locations

- 0.00 Linux Cluster
  - 0.00 zam008e1
    - 0.01 Process 0
    - 0.00 Process 1
    - 0.00 Process 2
    - 0.00 Process 3
  - 0.00 zam008e2
    - 0.00 Process 4
    - 0.00 Process 5

Exponent at the bottom

Percentage Mode

- All values percentages of the total execution time



CUBE: cx_8x1.cube

Values can become very small

Performance Metrics

- 0.0 Total
  - 80.4 Execution
    - 1.6 MPI
      - 0.0 Communication
        - 5.1 Collective
        - 7.2 P2P
          - 0.1 Late Receiver
          - 5.6 Late Sender
      - 0.0 IO
      - 0.0 Synchronization

Call Tree

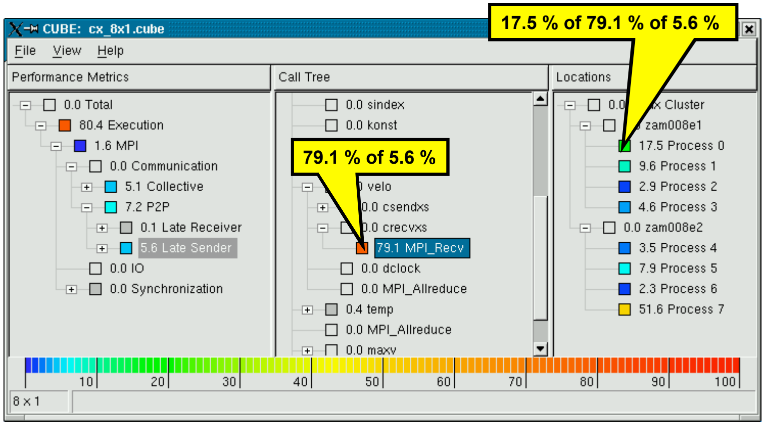
- 0.0 sindex
- 0.0 konst
- 1.1 bound
- 0.0 curr
- 0.0 velo
  - 0.0 csendxs
  - 0.0 crecvxs
    - 4.4 MPI_Recv
  - 0.0 dclock
  - 0.0 MPI_Allreduce
- 0.0 temp
- 0.0 MPI_Allreduce
- 0.0 maxv

Locations

- 0.0 Linux Cluster
  - 0.0 zam008e1
    - 0.8 Process 0
    - 0.4 Process 1
    - 0.1 Process 2
    - 0.2 Process 3
  - 0.0 zam008e2
    - 0.2 Process 4
    - 0.3 Process 5
    - 0.1 Process 6
    - 2.3 Process 7

Relative Percentage Mode

- All values percentages of the selection in the left neighbor tree

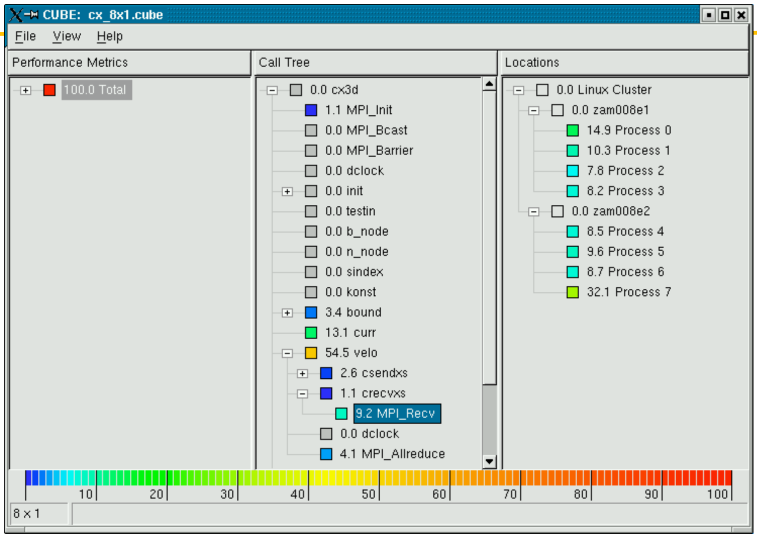


17.5 % of 79.1 % of 5.6 %

79.1 % of 5.6 %

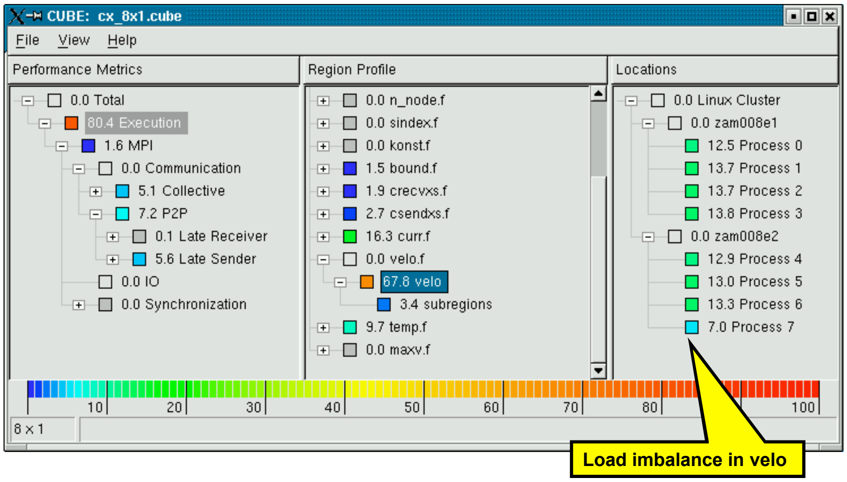
```
Performance Metrics
⊟ □ 0.0 Total
  ⊟ 🟥 80.4 Execution
    ⊟ 🟦 1.6 MPI
      ⊟ □ 0.0 Communication
        ⊞ 🟦 5.1 Collective
        ⊟ 🟦 7.2 P2P
          ⊞ □ 0.1 Late Receiver
          ⊞ 🟦 5.6 Late Sender
      □ 0.0 IO
      ⊞ □ 0.0 Synchronization

Call Tree
  □ 0.0 sindex
  □ 0.0 konst
⊟   velo
  ⊞   0.0 csendxs
  ⊟ □ 0.0 crecvxs
      🟥 79.1 MPI_Recv
    □ 0.0 dclock
    □ 0.0 MPI_Allreduce
⊞ □ 0.4 temp
  □ 0.0 MPI_Allreduce
⊞ □ 0.0 maxv

Locations
⊟ □ 0.0   x Cluster
  ⊟ □   zam008e1
      🟩 17.5 Process 0
      🟦 9.6 Process 1
      🟦 2.9 Process 2
      🟦 4.6 Process 3
  ⊟ □ 0.0 zam008e2
      🟦 3.5 Process 4
      🟦 7.9 Process 5
      🟦 2.3 Process 6
      🟨 51.6 Process 7
```

Call Tree View



```
Performance Metrics
⊞ 🟥 100.0 Total

Call Tree
⊟ □ 0.0 cx3d
    🟦 1.1 MPI_Init
    □ 0.0 MPI_Bcast
    □ 0.0 MPI_Barrier
    □ 0.0 dclock
  ⊞ □ 0.0 init
    □ 0.0 testin
    □ 0.0 b_node
    □ 0.0 n_node
    □ 0.0 sindex
    □ 0.0 konst
  ⊞ 🟦 3.4 bound
    🟩 13.1 curr
  ⊟ 🟨 54.5 velo
    ⊞ 🟦 2.6 csendxs
    ⊟ 🟦 1.1 crecvxs
        🟩 9.2 MPI_Recv
      □ 0.0 dclock
      🟦 4.1 MPI_Allreduce

Locations
⊟ □ 0.0 Linux Cluster
  ⊟ □ 0.0 zam008e1
      🟩 14.9 Process 0
      🟩 10.3 Process 1
      🟩 7.6 Process 2
      🟩 8.2 Process 3
  ⊟ □ 0.0 zam008e2
      🟩 8.5 Process 4
      🟩 9.6 Process 5
      🟩 8.7 Process 6
      🟩 32.1 Process 7
```

Region Profile View



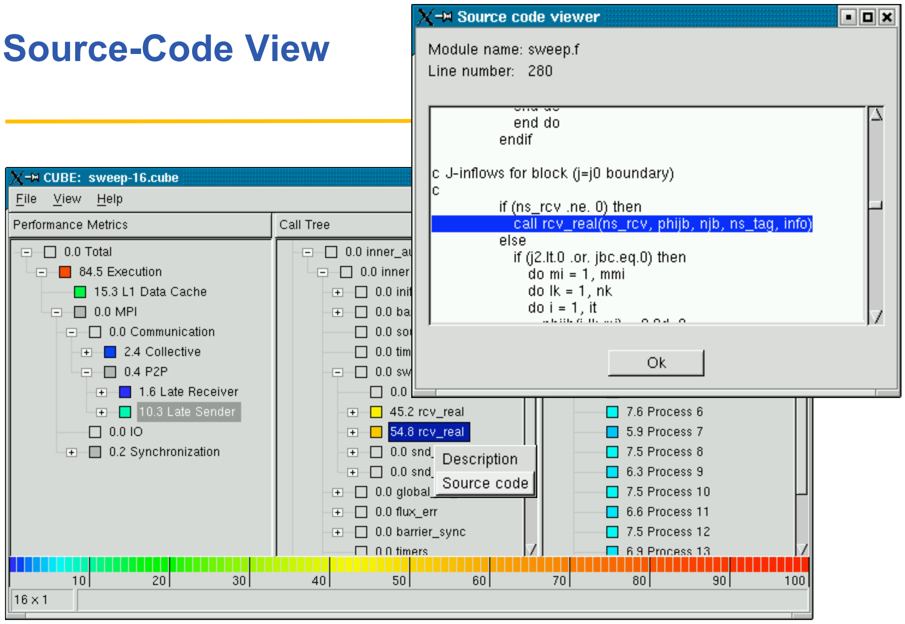
Performance Metrics:
- 0.0 Total
  - 80.4 Execution
    - 1.6 MPI
      - 0.0 Communication
        - 5.1 Collective
        - 7.2 P2P
          - 0.1 Late Receiver
          - 5.6 Late Sender
      - 0.0 IO
      - 0.0 Synchronization

Region Profile:
- 0.0 n_node.f
- 0.0 sindex.f
- 0.0 konst.f
- 1.5 bound.f
- 1.9 crecvxs.f
- 2.7 csendxs.f
- 16.3 curr.f
- 0.0 velo.f
  - 67.8 velo
    - 3.4 subregions
- 9.7 temp.f
- 0.0 maxv.f

Locations:
- 0.0 Linux Cluster
  - 0.0 zam008e1
    - 12.5 Process 0
    - 13.7 Process 1
    - 13.7 Process 2
    - 13.8 Process 3
  - 0.0 zam008e2
    - 12.9 Process 4
    - 13.0 Process 5
    - 13.3 Process 6
    - 7.0 Process 7

Load imbalance in velo

Source-Code View



Module name: sweep.f
Line number: 280

```
           end do
         endif

c J-inflows for block (j=j0 boundary)
c
         if (ns_rcv .ne. 0) then
           call rcv_real(ns_rcv, phijb, njb, ns_tag, info)
         else
           if (j2.lt.0 .or. jbc.eq.0) then
             do mi = 1, mmi
             do lk = 1, nk
             do i = 1, it
```

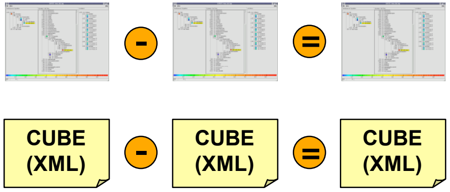
Performance Metrics:
- 0.0 Total
  - 84.5 Execution
    - 15.3 L1 Data Cache
    - 0.0 MPI
      - 0.0 Communication
        - 2.4 Collective
        - 0.4 P2P
          - 1.6 Late Receiver
          - 10.3 Late Sender
      - 0.0 IO
      - 0.2 Synchronization

Call Tree:
- 0.0 inner_au
  - 0.0 inner
    - 0.0 init
    - 0.0 ba
    - 0.0 so
    - 0.0 tim
    - 0.0 sw
      - 0.0
      - 45.2 rcv_real
      - 54.8 rcv_real
      - 0.0 snd_
      - 0.0 snd_
    - 0.0 global
    - 0.0 flux_err
    - 0.0 barrier_sync
    - 0.0 timers

- 7.6 Process 6
- 5.9 Process 7
- 7.5 Process 8
- 6.3 Process 9
- 7.5 Process 10
- 6.6 Process 11
- 7.5 Process 12
- 6.9 Process 13

Performance Algebra

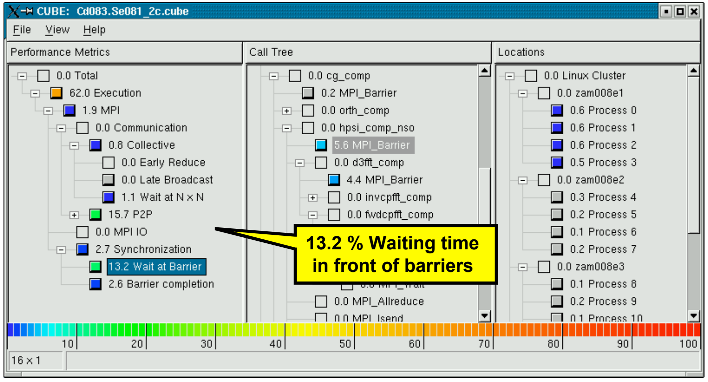
- Cross-experiment analysis
  - Different execution configuration
  - Different measurement tools
  - Different random errors



- Arithmetic operations on CUBE instances
  - Difference, merge, mean
  - Obtain CUBE instance as result
  - Display it like ordinary CUBE instance

Nano-Particle Simulation PESCAN

- Application Lawrence Berkeley National Lab
- Numerous barriers to avoid buffer overflow when using large processor counts – not needed for smaller counts

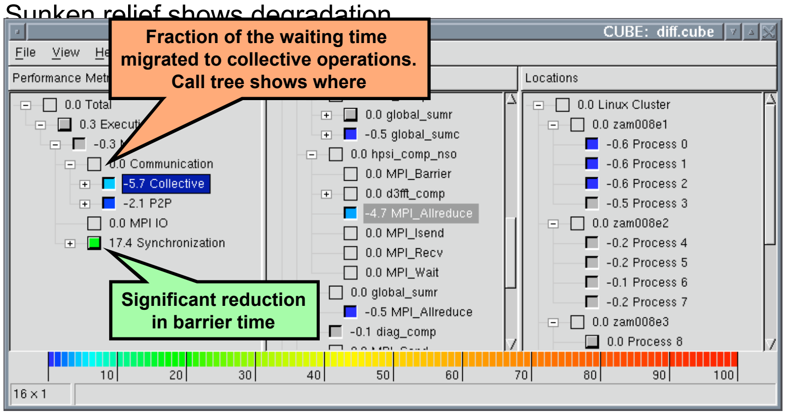


```
Performance Metrics
0.0 Total
  62.0 Execution
    1.9 MPI
      0.0 Communication
        0.8 Collective
          0.0 Early Reduce
          0.0 Late Broadcast
          1.1 Wait at N x N
        15.7 P2P
      0.0 MPI IO
      2.7 Synchronization
        13.2 Wait at Barrier
        2.6 Barrier completion

Call Tree
0.0 cg_comp
  0.2 MPI_Barrier
  0.0 orth_comp
  0.0 hpsi_comp_nso
    5.6 MPI_Barrier
    0.0 d3fft_comp
      4.4 MPI_Barrier
      0.0 invcpfft_comp
      0.0 fwdcpfft_comp
    0.0 MPI_Allreduce
    0.0 MPI_Isend

Locations
0.0 Linux Cluster
  0.0 zam008e1
    0.6 Process 0
    0.6 Process 1
    0.6 Process 2
    0.5 Process 3
  0.0 zam008e2
    0.3 Process 4
    0.2 Process 5
    0.1 Process 6
    0.2 Process 7
  0.0 zam008e3
    0.1 Process 8
    0.2 Process 9
    0.1 Process 10
```

13.2 % Waiting time in front of barriers

(Re)moving Waiting Times

- Difference between before / after barrier removal
- Raised relief shows improvement
- Sunken relief shows degradation



CUBE: diff.cube

```
Performance Metrics
⊟ □ 0.0 Total
  ⊟ □ 0.3 Executi
    ⊟ □ -0.3
      ⊟ □ 0.0 Communication
        ⊞ □ -5.7 Collective
        ⊞ □ -2.1 P2P
      □ 0.0 MPI IO
      ⊞ □ 17.4 Synchronization
```

```
  ⊞ □ 0.0 global_sumr
  ⊞ □ -0.5 global_sumc
⊟ □ 0.0 hpsi_comp_nso
    □ 0.0 MPI_Barrier
  ⊞ □ 0.0 d3fft_comp
    □ -4.7 MPI_Allreduce
    □ 0.0 MPI_Isend
    □ 0.0 MPI_Recv
    □ 0.0 MPI_Wait
  □ 0.0 global_sumr
    □ -0.5 MPI_Allreduce
  □ -0.1 diag_comp
```

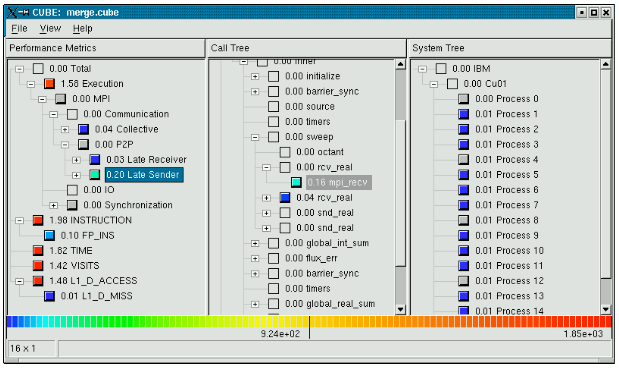
```
Locations
⊟ □ 0.0 Linux Cluster
  ⊟ □ 0.0 zam008e1
      □ -0.6 Process 0
      □ -0.6 Process 1
      □ -0.6 Process 2
      □ -0.5 Process 3
  ⊟ □ 0.0 zam008e2
      □ -0.2 Process 4
      □ -0.2 Process 5
      □ -0.1 Process 6
      □ -0.2 Process 7
  ⊟ □ 0.0 zam008e3
      □ 0.0 Process 8
```

Fraction of the waiting time migrated to collective operations. Call tree shows where

Significant reduction in barrier time

Integrating Performance Data From Multiple Sources

- Integrating trace data with profile data
- Integrating data that cannot be generated simultaneously
  - L1 cache misses and floating-point instructions on Power4
- KOJAK output merged with 2 CONE outputs



CUBE: merge.cube

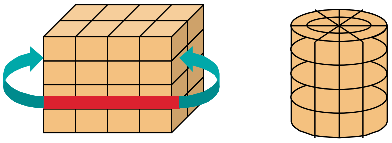
```
Performance Metrics
⊟ □ 0.00 Total
  ⊟ □ 1.58 Execution
    ⊟ □ 0.00 MPI
      ⊟ □ 0.00 Communication
        ⊞ □ 0.04 Collective
        ⊟ □ 0.00 P2P
          ⊞ □ 0.03 Late Receiver
          ⊞ □ 0.20 Late Sender
      □ 0.00 IO
      ⊞ □ 0.00 Synchronization
⊟ □ 1.98 INSTRUCTION
    □ 0.10 FP_INS
  □ 1.82 TIME
  □ 1.42 VISITS
⊟ □ 1.48 L1_D_ACCESS
    □ 0.01 L1_D_MISS
```

```
Call Tree
⊟ □ 0.00 inner
  ⊞ □ 0.00 initialize
  ⊞ □ 0.00 barrier_sync
    □ 0.00 source
    □ 0.00 timers
  ⊟ □ 0.00 sweep
      □ 0.00 octant
    ⊟ □ 0.00 rcv_real
        □ 0.16 mpi_recv
    ⊞ □ 0.04 rcv_real
    ⊞ □ 0.00 snd_real
    ⊞ □ 0.00 snd_real
  ⊞ □ 0.00 global_int_sum
  ⊞ □ 0.00 flux_err
  ⊞ □ 0.00 barrier_sync
    □ 0.00 timers
  ⊞ □ 0.00 global_real_sum
```

```
System Tree
⊟ □ 0.00 IBM
  ⊟ □ 0.00 Cu01
      □ 0.00 Process 0
      □ 0.01 Process 1
      □ 0.01 Process 2
      □ 0.01 Process 3
      □ 0.01 Process 4
      □ 0.01 Process 5
      □ 0.01 Process 6
      □ 0.01 Process 7
      □ 0.01 Process 8
      □ 0.01 Process 9
      □ 0.01 Process 10
      □ 0.01 Process 11
      □ 0.01 Process 12
      □ 0.01 Process 13
      □ 0.01 Process 14
```

Topologies

- Define adjacency relationships among system resources
- Virtual topologies
  - Mapping of processes/threads onto application domain
  - Parallel algorithms often parameterized in terms of topology
- Physical topologies
  - Network structure
- Can be specified as a graph
- Very common case: Cartesian topologies

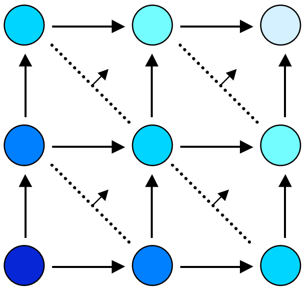


Topology analysis in KOJAK

- Idea: map performance data onto topology
  - Virtual or physical
- Record topological information in event trace
- Detect higher-level events related to the parallel algorithm
- Link occurrence of patterns to such higher-level events
- Visually expose correlations of performance problems with topological characteristics of affected processes

Analysis of wave-front processes

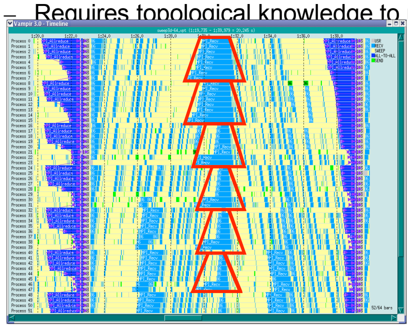
- Parallelization scheme used for particle transport problems
- Example: ASCI benchmark SWEEP3D
  - Three-dimensional domain (i,j,k)
  - Two-dimensional domain decomposition (i,j)

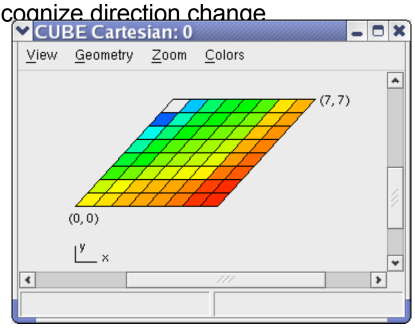


```
DO octants
  DO angles in octant
    DO k planes
      ! block i-inflows
      IF neighbor(E/W) MPI_RECV(E/W)
      ! block j-inflows
      IF neighbor(N/S) MPI_RECV(N/S)
         ... compute grid cell ...
      ! block i-outflows
      IF neighbor(E/W) MPI_SEND(E/W)
      ! block j-outflows
      IF neighbor(N/S) MPI_SEND(N/S)
    END DO k planes
  END DO angles in octant
END DO octants
```

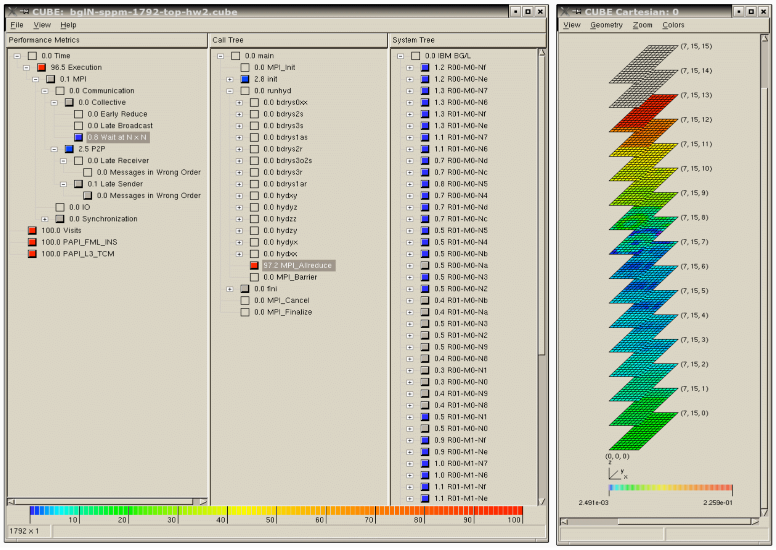
Pipeline refill

- Wave-fronts from different directions
- Limited parallelism upon pipeline refill
- Four new late-sender patterns
  - Refill from NW, NE, SE, SW
  - Requires topological knowledge to recognize direction change





Waiting in n-to-n on BG/L (1792 CPUs)



Performance Metrics

```
0.0 Time
  96.5 Execution
    0.1 MPI
      0.0 Communication
        0.0 Collective
          0.0 Early Reduce
          0.0 Late Broadcast
          0.8 Wait at N x N
        2.5 P2P
          0.0 Late Receiver
            0.0 Messages in Wrong Order
          0.1 Late Sender
            0.0 Messages in Wrong Order
      0.0 IO
      0.0 Synchronization
100.0 Visits
100.0 PAPI_FML_INS
100.0 PAPI_L3_TCM
```

Call Tree

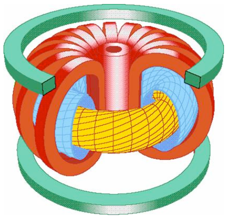
```
0.0 main
  0.0 MPI_Init
  2.8 init
  0.0 runhyd
    0.0 bdrys0xx
    0.0 bdrys2s
    0.0 bdrys3s
    0.0 bdrys1as
    0.0 bdrys2r
    0.0 bdrys3o2s
    0.0 bdrys3r
    0.0 bdrys1ar
    0.0 hydxy
    0.0 hydyz
    0.0 hydzz
    0.0 hydzy
    0.0 hydyx
    0.0 hydxx
    97.2 MPI_Allreduce
    0.0 MPI_Barrier
  0.0 fini
  0.0 MPI_Cancel
  0.0 MPI_Finalize
```

System Tree

```
0.0 IBM BG/L
  1.2 R00-M0-Nf
  1.2 R00-M0-Ne
  1.3 R00-M0-N7
  1.3 R00-M0-N6
  1.3 R01-M0-Nf
  1.3 R01-M0-Ne
  1.1 R01-M0-N7
  1.1 R01-M0-N6
  0.7 R00-M0-Nd
  0.7 R00-M0-Nc
  0.8 R00-M0-N5
  0.7 R00-M0-N4
  0.7 R01-M0-Nd
  0.7 R01-M0-Nc
  0.5 R01-M0-N5
  0.5 R01-M0-N4
  0.5 R00-M0-Nb
  0.5 R00-M0-Na
  0.5 R00-M0-N3
  0.5 R00-M0-N2
  0.4 R01-M0-Nb
  0.4 R01-M0-Na
  0.5 R01-M0-N3
  0.5 R01-M0-N2
  0.5 R00-M0-N9
  0.4 R00-M0-N8
  0.3 R00-M0-N1
  0.3 R00-M0-N0
  0.4 R01-M0-N9
  0.4 R01-M0-N8
  0.5 R01-M0-N1
  0.5 R01-M0-N0
  0.9 R00-M1-Nf
  0.9 R00-M1-Ne
  1.0 R00-M1-N7
  1.0 R00-M1-N6
  1.1 R01-M1-Nf
  1.1 R01-M1-Ne
```

SciDAC application GYRO

- SciDAC (Scientific Discovery through Advanced Computing)
  - Develop the software and hardware infrastructure needed to use tera-scale computers to advance DOE research programs in
    - Basic energy sciences, biological and environmental research, fusion energy sciences, and high-energy and nuclear physics.
- PERC (Performance Evaluation Research Center)
  - Develop a science for understanding performance of scientific applications on high-end computer systems
  - Develop engineering strategies for g performance on these systems
- Application GYRO
  - 5D gyrokinetic Maxwell solver
  - Simulates turbulences in magnetically confined plasmas using message passing



Source: Max Planck Institut für Plasma Physik, Germany

Trace-size reduction with call-path profiling

- Instrumentation of user functions necessary to identify call path of performance problem
- Non-discriminate instrumentation of user functions can lead to significant trace-file enlargement
- Example: estimated trace-file size for fully instrumented run of GYRO B1-std benchmark on 32 CPUs > 100 GB
- Effective reduction possible using TAU / PDT
  - Generate call path profile
  - Instrument only those functions that call communication or synchronization routines (directly or indirectly)
- Result

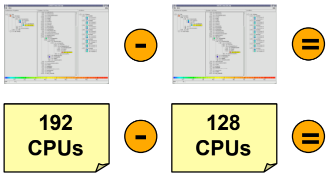
#CPU	32	64	128	192
Size (MB)	94	188	375	562

UNIVERSITY OF OREGON

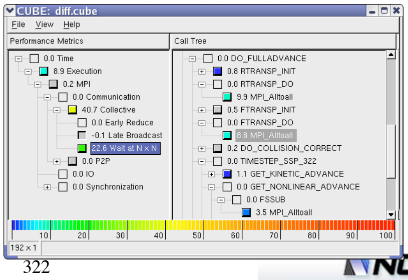
NCSA

Scalability analysis of GYRO on SGI Altix

- Drop off in parallel efficiency when #CPUs exceeds 128
- 192 CPUs
  - 49 % of execution time in collective communication
  - 21 % of execution time wait time in n-to-n operations
- Raising #CPU from 128 to 192 increases accumulated execution time (CPU sec) by 72 %
- Performance algebra shows composition of differences



192 CPUs - 128 CPUs =



```
0.0 Time
  8.9 Execution
    0.2 MPI
      0.0 Communication
        40.7 Collective
          0.0 Early Reduce
          -0.1 Late Broadcast
          22.6 Wait at N x N
        0.0 P2P
      0.0 IO
      0.0 Synchronization
```

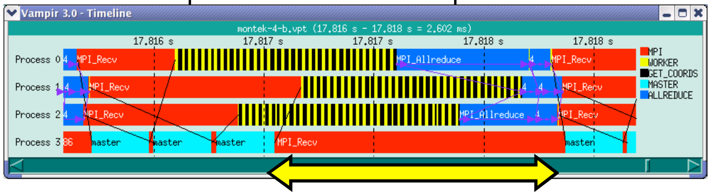
```
0.0 DO_FULLADVANCE
  0.8 RTRANSP_INIT
  0.0 RTRANSP_DO
    9.9 MPI_Alltoall
  0.5 FTRANSP_INIT
  0.0 FTRANSP_DO
    8.8 MPI_Alltoall
  0.2 DO_COLLISION_CORRECT
  0.0 TIMESTEP_SSP_322
    1.1 GET_KINETIC_ADVANCE
    0.0 GET_NONLINEAR_ADVANCE
      0.0 FSSUB
        3.5 MPI_Alltoall
```

UNIVERSITY OF OREGON

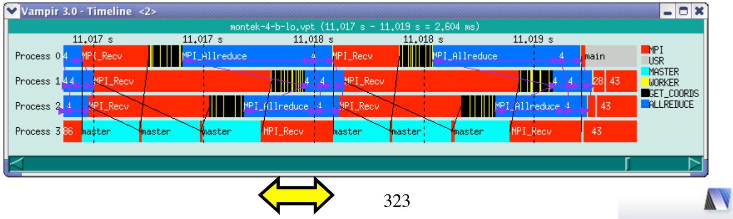
NCSA

Instrumentation alters performance behavior

- Execution of extra code, consumption of resources
  - Intrusion: quantitative dilation of execution time
  - Perturbation: qualitative alteration of performance behavior



- Off-line perturbation compensation to approximate original behavior (with University of Oregon)



To use KOJAK with TAU

- Choose a TAU stub makefile with a -epilog in its name

% setenv TAU_MAKEFILE /usr/pkgs/tau/<arch>/lib/Makefile.tau-mpi-pdt-epilog-trace

- Change CC to tau_cc.sh, F90 to tau_f90.sh in your Makefile
- Build and execute the program

% make; mpirun -np 6 sweep3d.mpi

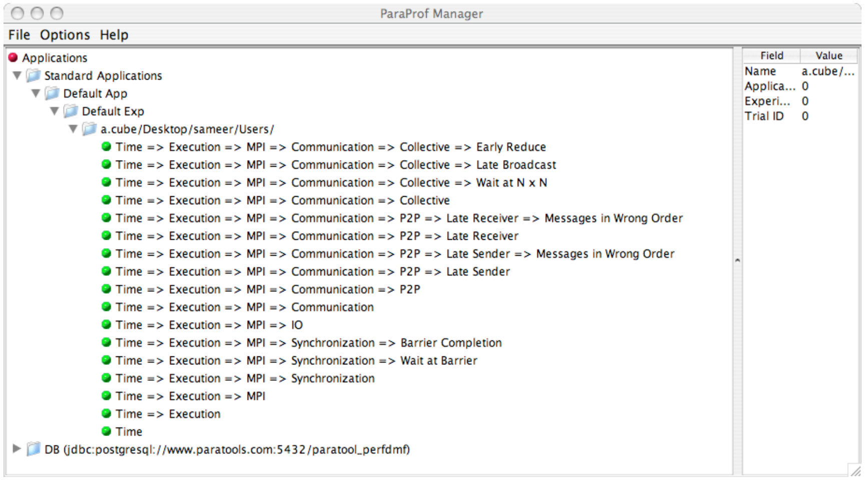
- Run the Expert tool on the generated a.elg merged trace file

% expert a.elg

- Load the generated a.cube file in Paraprof and click on metrics of interest

% paraprof a.cube

ParaProf: Performance Bottlenecks



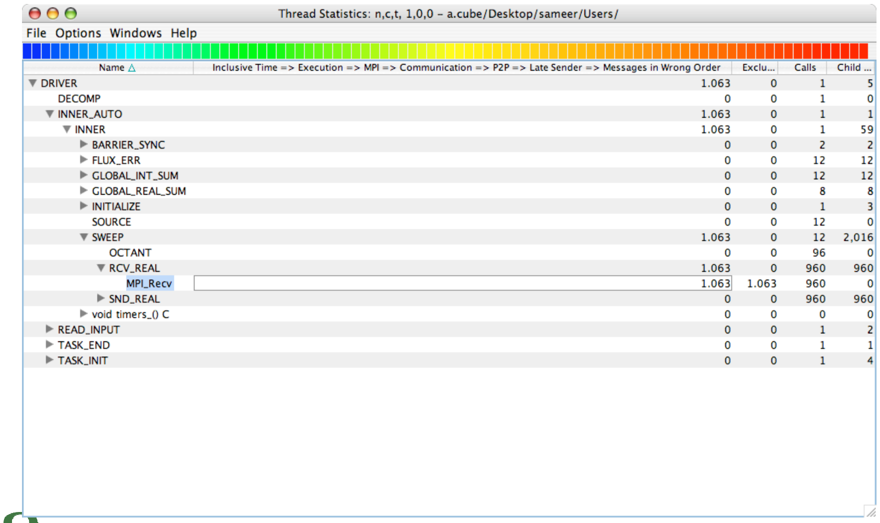
ParaProf Manager

File Options Help

- Applications
  - Standard Applications
    - Default App
      - Default Exp
        - a.cube/Desktop/sameer/Users/
          - Time => Execution => MPI => Communication => Collective => Early Reduce
          - Time => Execution => MPI => Communication => Collective => Late Broadcast
          - Time => Execution => MPI => Communication => Collective => Wait at N x N
          - Time => Execution => MPI => Communication => Collective
          - Time => Execution => MPI => Communication => P2P => Late Receiver => Messages in Wrong Order
          - Time => Execution => MPI => Communication => P2P => Late Receiver
          - Time => Execution => MPI => Communication => P2P => Late Sender => Messages in Wrong Order
          - Time => Execution => MPI => Communication => P2P => Late Sender
          - Time => Execution => MPI => Communication => P2P
          - Time => Execution => MPI => Communication
          - Time => Execution => MPI => IO
          - Time => Execution => MPI => Synchronization => Barrier Completion
          - Time => Execution => MPI => Synchronization => Wait at Barrier
          - Time => Execution => MPI => Synchronization
          - Time => Execution => MPI
          - Time => Execution
          - Time
  - DB (jdbc:postgresql://www.paratools.com:5432/paratool_perfdmf)

Field	Value
Name	a.cube/...
Applica...	0
Experi...	0
Trial ID	0

ParaProf: Time spent in late sender bottleneck

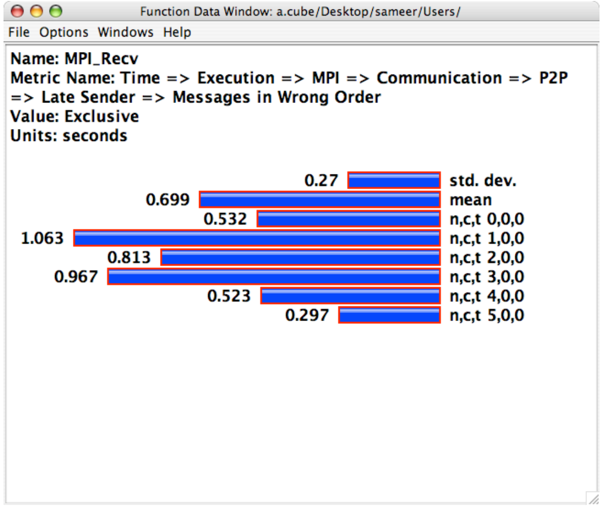


Thread Statistics: n,c,t, 1,0,0 – a.cube/Desktop/sameer/Users/

File Options Windows Help

Name △	Inclusive Time => Execution => MPI => Communication => P2P => Late Sender => Messages in Wrong Order	Exclu...	Calls	Child ...
▼ DRIVER	1.063	0	1	5
DECOMP	0	0	1	0
▼ INNER_AUTO	1.063	0	1	1
▼ INNER	1.063	0	1	59
▶ BARRIER_SYNC	0	0	2	2
▶ FLUX_ERR	0	0	12	12
▶ GLOBAL_INT_SUM	0	0	12	12
▶ GLOBAL_REAL_SUM	0	0	8	8
▶ INITIALIZE	0	0	1	3
SOURCE	0	0	12	0
▼ SWEEP	1.063	0	12	2,016
OCTANT	0	0	96	0
▼ RCV_REAL	1.063	0	960	960
MPI_Recv	1.063	1.063	960	0
▶ SND_REAL	0	0	960	960
▶ void timers_() C	0	0	0	0
▶ READ_INPUT	0	0	1	2
▶ TASK_END	0	0	1	1
▶ TASK_INIT	0	0	1	4

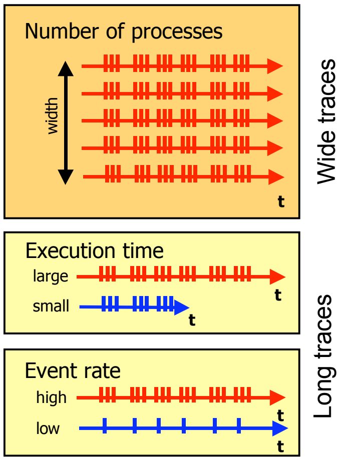
ParaProf: Time spent in late sender bottleneck

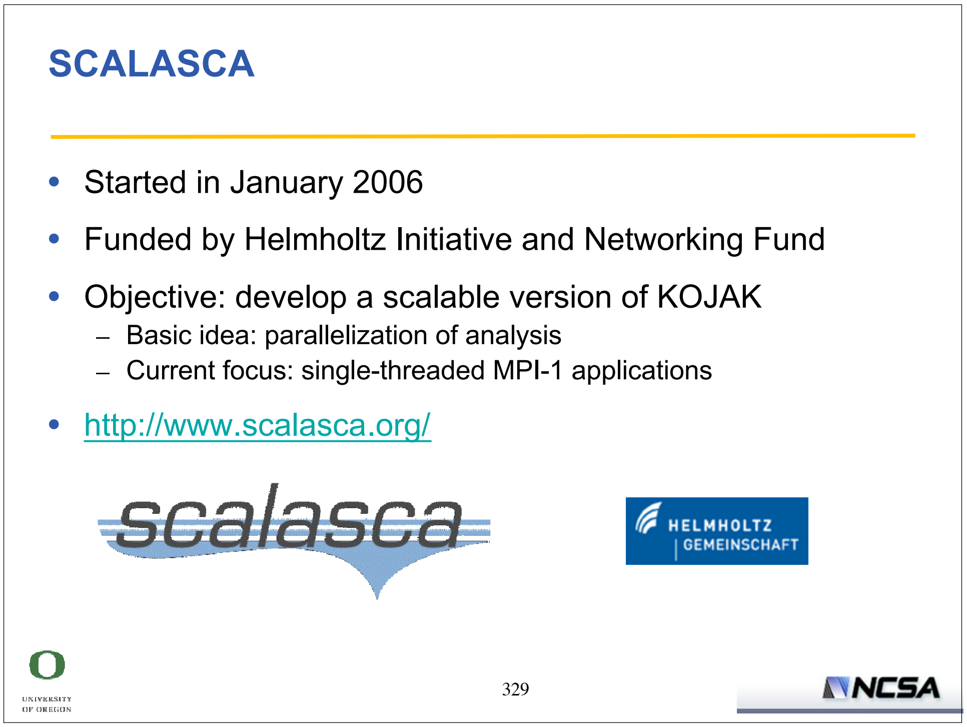


Name: MPI_Recv
Metric Name: Time => Execution => MPI => Communication => P2P => Late Sender => Messages in Wrong Order
Value: Exclusive
Units: seconds

Trace size limits scalability

- Serially analyzing a single global trace file does not scale to 1000s of processors
- Main memory might be insufficient to store context of current event
- Amount of trace data might not fit into single file



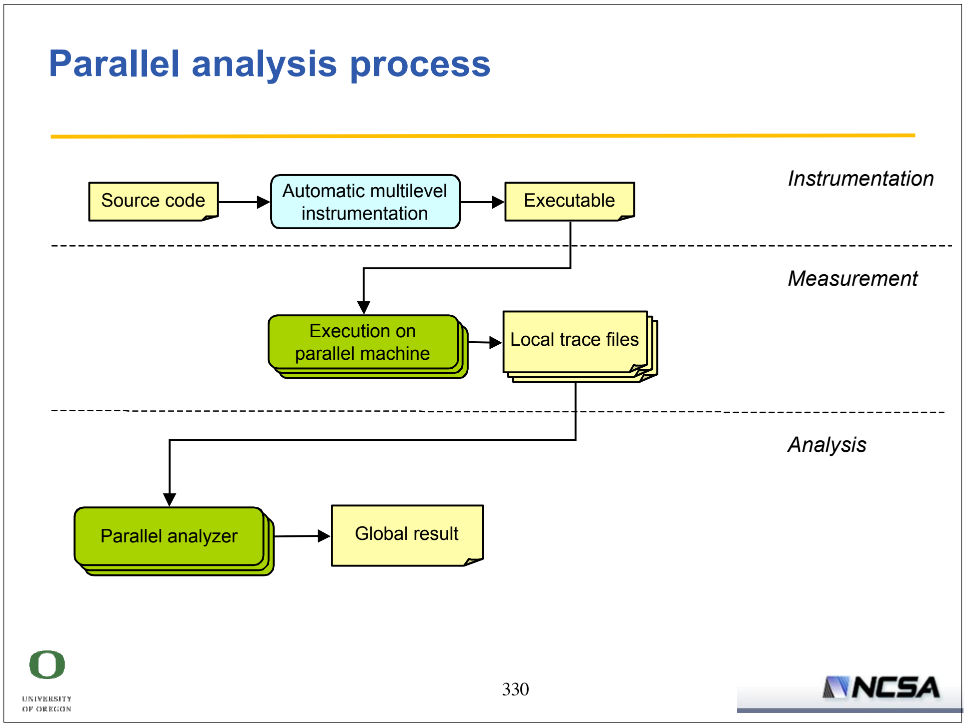


SCALASCA

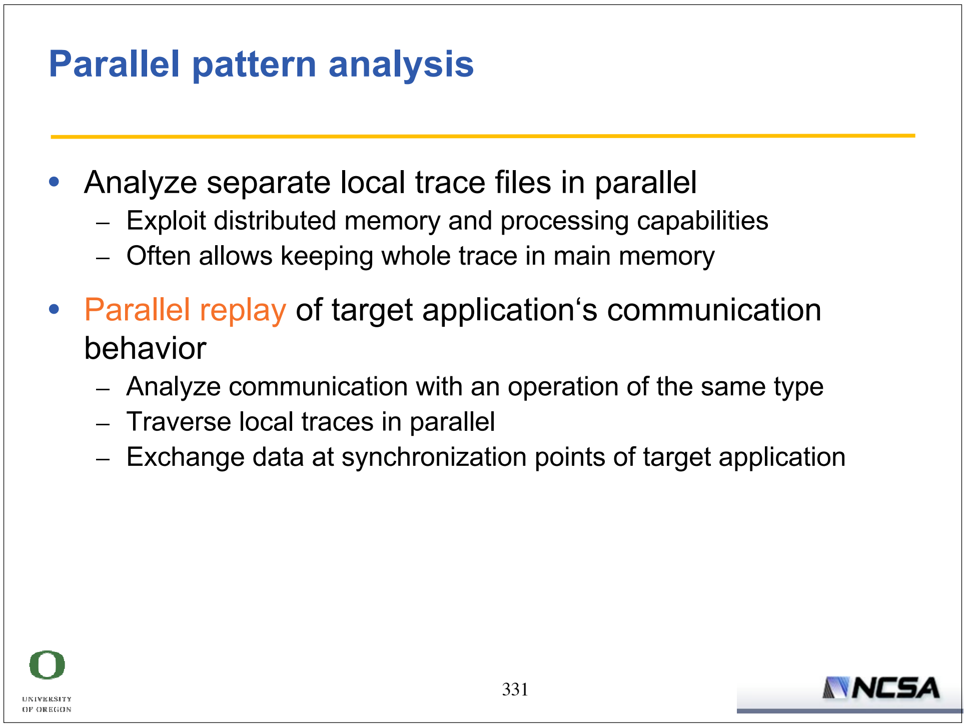
- Started in January 2006
- Funded by Helmholtz Initiative and Networking Fund
- Objective: develop a scalable version of KOJAK
  - Basic idea: parallelization of analysis
  - Current focus: single-threaded MPI-1 applications
- http://www.scalasca.org/

scalasca

HELMHOLTZ GEMEINSCHAFT

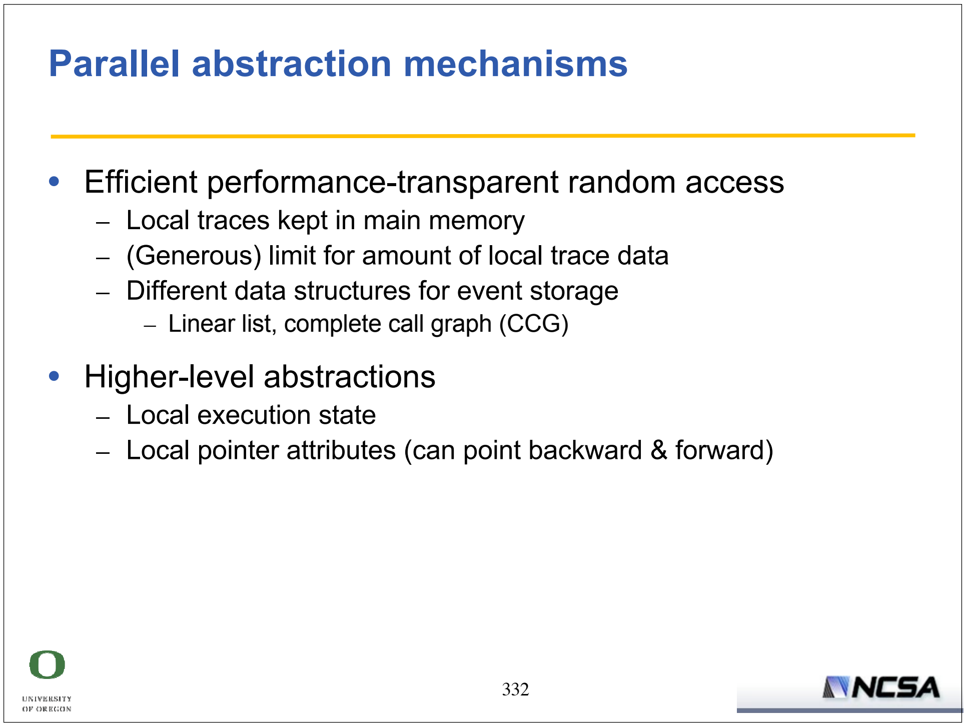


Parallel analysis process



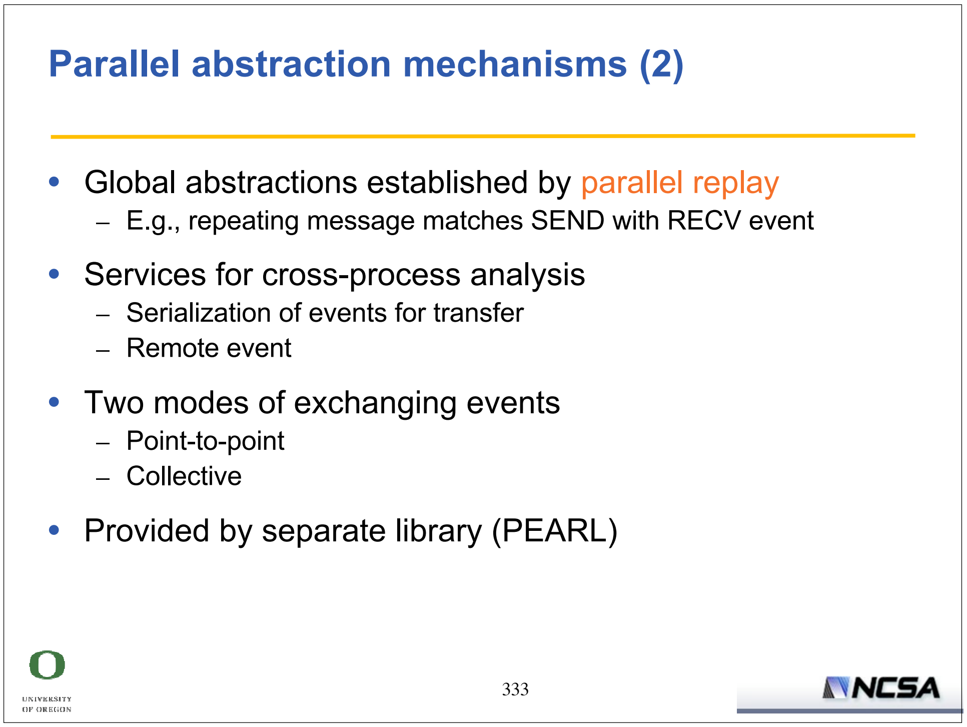
Parallel pattern analysis

- Analyze separate local trace files in parallel
  - Exploit distributed memory and processing capabilities
  - Often allows keeping whole trace in main memory
- Parallel replay of target application's communication behavior
  - Analyze communication with an operation of the same type
  - Traverse local traces in parallel
  - Exchange data at synchronization points of target application



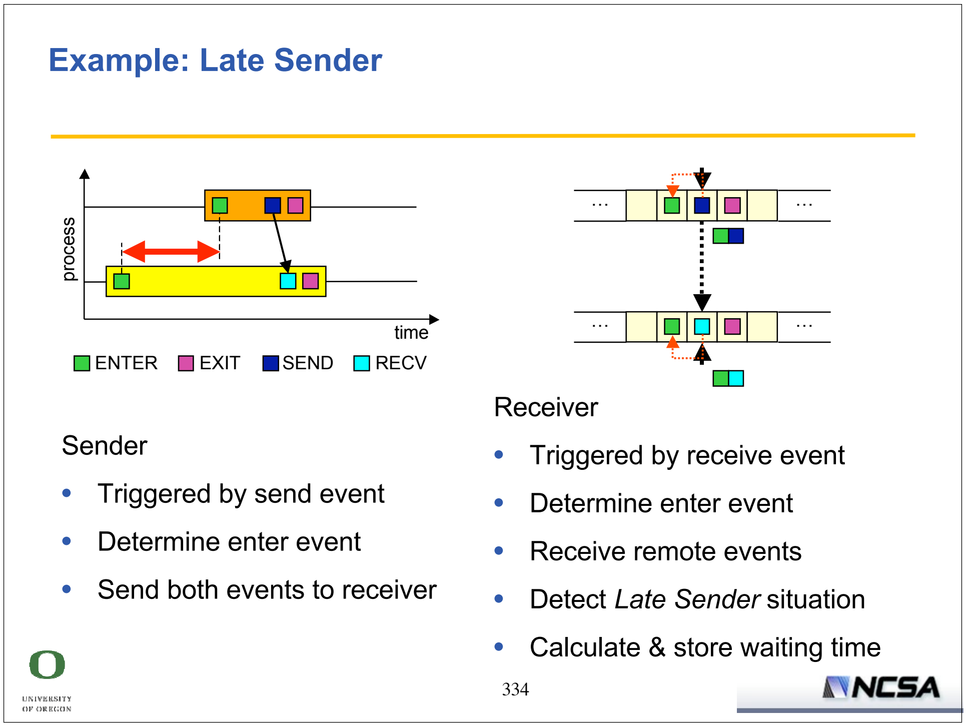
Parallel abstraction mechanisms

- Efficient performance-transparent random access
  - Local traces kept in main memory
  - (Generous) limit for amount of local trace data
  - Different data structures for event storage
    - Linear list, complete call graph (CCG)
- Higher-level abstractions
  - Local execution state
  - Local pointer attributes (can point backward & forward)



Parallel abstraction mechanisms (2)

- Global abstractions established by parallel replay
  - E.g., repeating message matches SEND with RECV event
- Services for cross-process analysis
  - Serialization of events for transfer
  - Remote event
- Two modes of exchanging events
  - Point-to-point
  - Collective
- Provided by separate library (PEARL)



Example: Late Sender

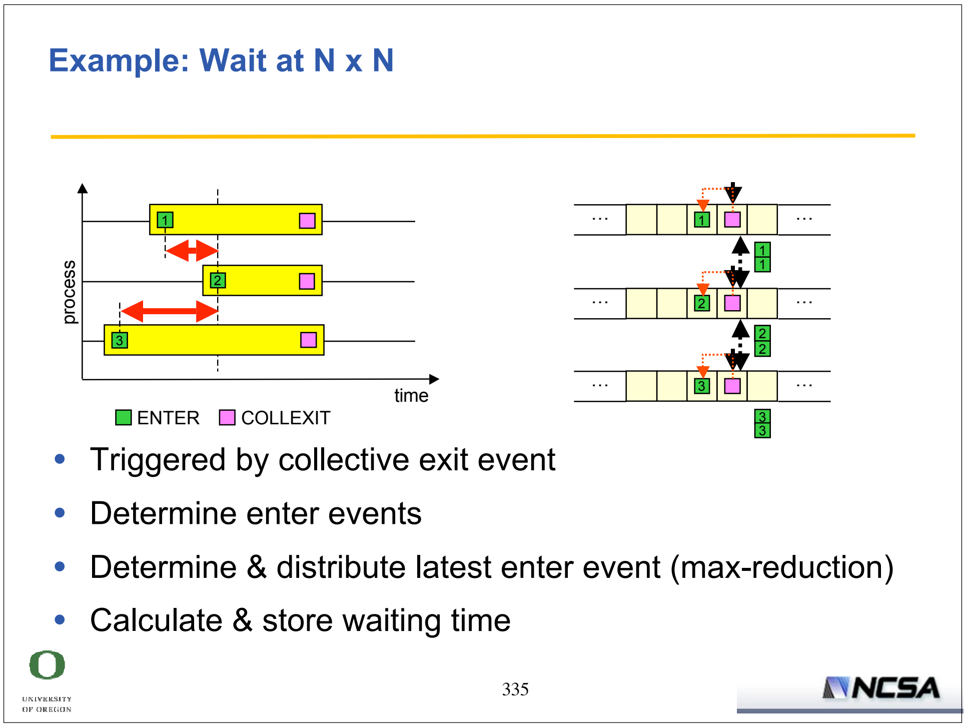
🟩 ENTER 🟪 EXIT 🟦 SEND ▢ RECV

Sender

- Triggered by send event
- Determine enter event
- Send both events to receiver

Receiver

- Triggered by receive event
- Determine enter event
- Receive remote events
- Detect *Late Sender* situation
- Calculate & store waiting time



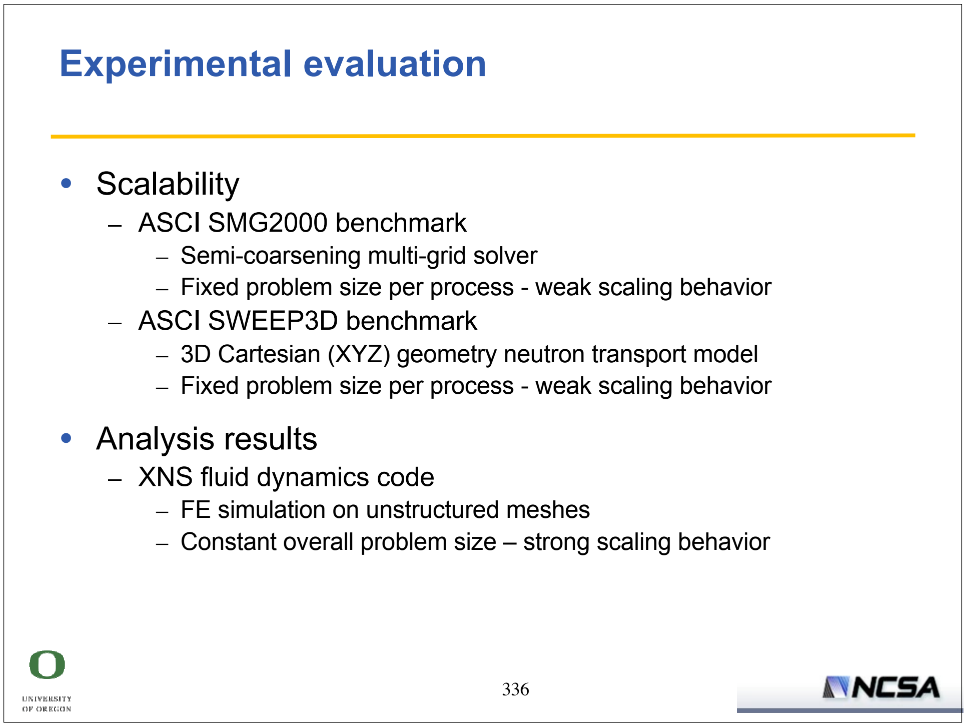
Example: Wait at N x N

🟩 ENTER 🟪 COLLEXIT

- Triggered by collective exit event
- Determine enter events
- Determine & distribute latest enter event (max-reduction)
- Calculate & store waiting time

UNIVERSITY OF OREGON

NCSA



Experimental evaluation

- Scalability
  - ASCI SMG2000 benchmark
    - Semi-coarsening multi-grid solver
    - Fixed problem size per process - weak scaling behavior
  - ASCI SWEEP3D benchmark
    - 3D Cartesian (XYZ) geometry neutron transport model
    - Fixed problem size per process - weak scaling behavior
- Analysis results
  - XNS fluid dynamics code
    - FE simulation on unstructured meshes
    - Constant overall problem size – strong scaling behavior

UNIVERSITY OF OREGON

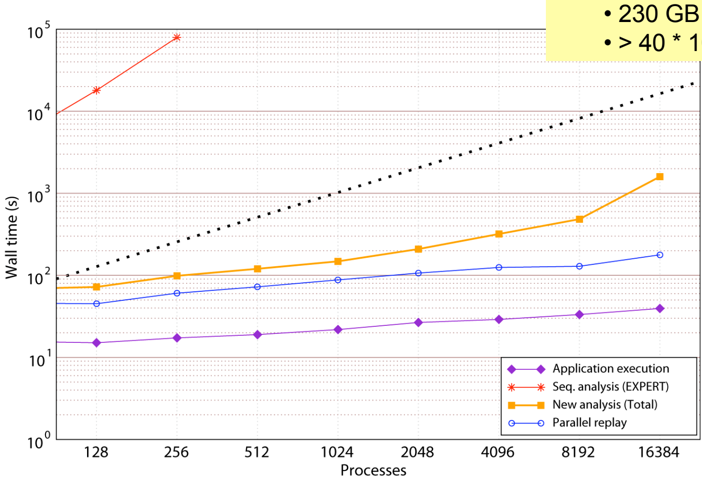
NCSA

Test platform

BlueGene/L (JUBL) in Jülich

- 8 Racks with 8192 dual-core nodes
- 288 I/O nodes

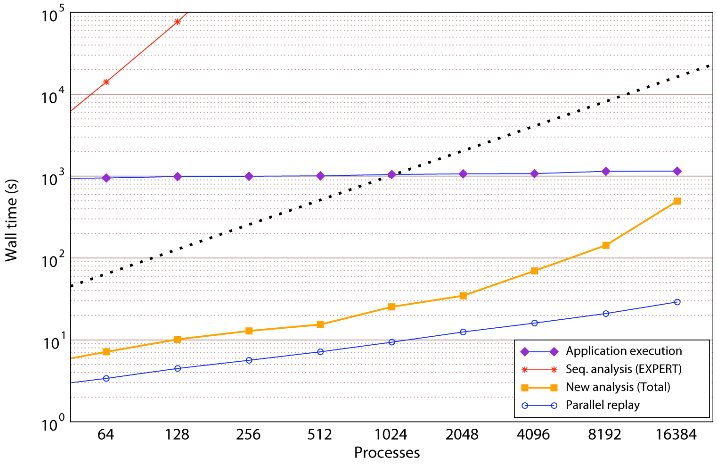
SMG2000



SMG2000 on 16,364 CPUs

- 230 GB trace data
- > 40 * 10^9 events

SWEEP3D

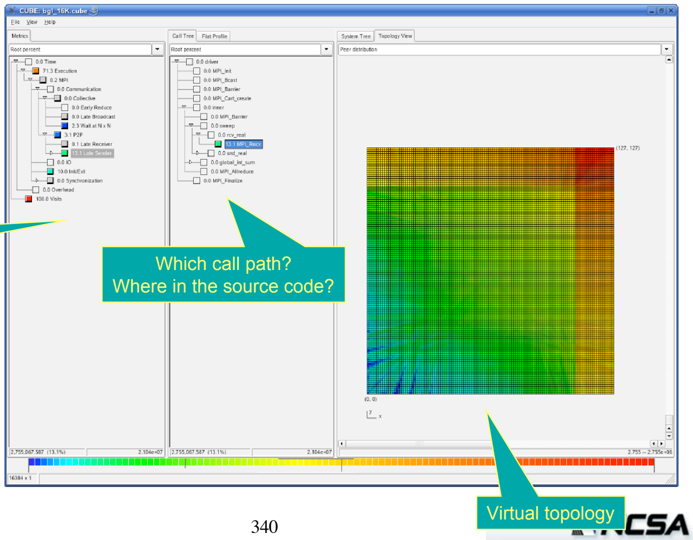


UNIVERSITY OF OREGON

NCSA

SWEEP3D

Late Sender
16K CPUs



Root percent

```
0.0 Time
  71.3 Execution
    0.2 MPI
      0.0 Communication
        0.0 Collective
          0.0 Early Reduce
          0.0 Late Broadcast
          2.3 Wait at N x N
        3.1 P2P
          0.1 Late Receiver
          13.1 Late Sender
      0.0 IO
      10.0 Init/Exit
      0.0 Synchronization
  0.0 Overhead
100.0 Visits
```

Root percent

```
0.0 driver
  0.0 MPI_Init
  0.0 MPI_Bcast
  0.0 MPI_Barrier
  0.0 MPI_Cart_create
  0.0 inner
    0.0 MPI_Barrier
    0.0 sweep
      0.0 rcv_real
        13.1 MPI_Recv
      0.0 snd_real
    0.0 global_int_sum
    0.0 MPI_Allreduce
  0.0 MPI_Finalize
```

Peer distribution

```
(127, 127)
(0, 0)
2,755,067.587 (13.1%)    2.104e+07
2,755,067.587 (13.1%)    2.104e+07
2.755 -- 2.755e+06
16384 x 1
```

Which type of problem?

Which call path? Where in the source code?

Virtual topology

UNIVERSITY OF OREGON

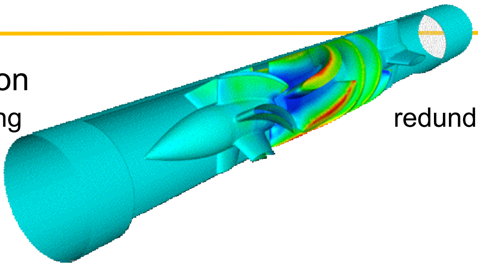
NCSA

XNS CFD application

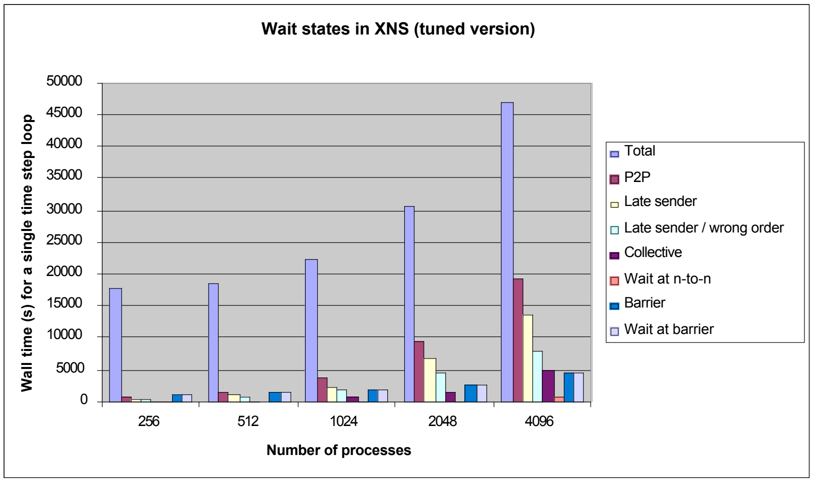
- Academic computational fluid dynamics code for simulation of unsteady flows
  - Developed by group of Marek Behr, Computational Analysis of Technical Systems, RWTH Aachen University
  - Exploits finite-element techniques, unstructured 3D meshes, iterative solution strategies
  - >40,000 lines of Fortran90
  - Portable parallel implementation based on MPI
- Focus on solver phase (i.e., ignoring I/O)

DeBakey blood pump test case

- Consider tuned version
  - Obtained by eliminating redundant messages
  - Scalability of original version limited <1024
  - Initial mpiP profiling
    - Dominated by growing MPI communication
    - Average message size diminishing (<16 bytes)
    - Growing numbers of zero-sized communications
  - Detailed SCALASCA trace analysis
    - Investigated message sizes & distributions
    - 96% of transfers redundant @1024 processes!



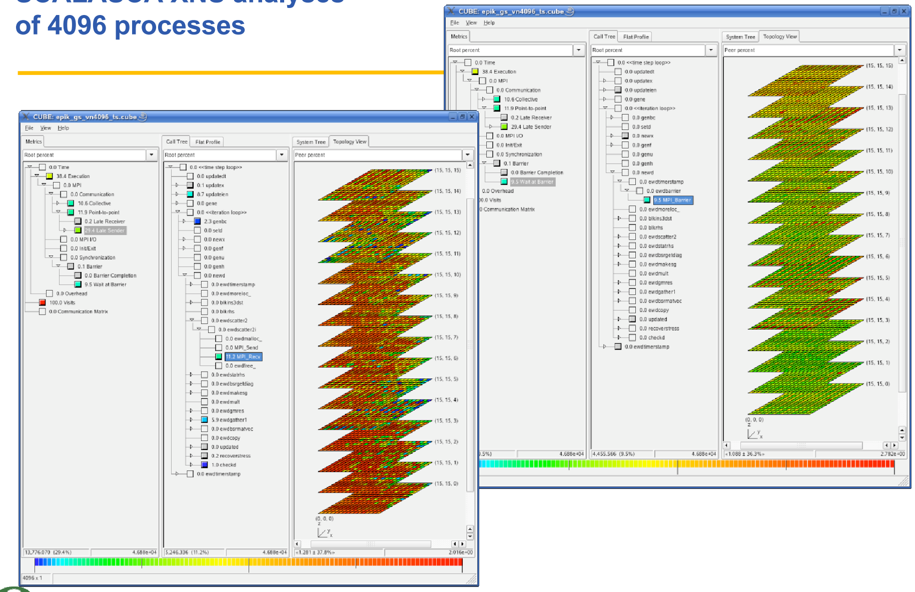
SCALASCA: Wait states in tuned version of XNS



UNIVERSITY OF OREGON

NCSA

SCALASCA XNS analyses of 4096 processes



0.0 Time
38.4 Execution
0.0 MPI
0.0 Communication
10.6 Collective
11.9 Point-to-point
0.2 Late Receiver
29.4 Late Sender
0.0 MPI I/O
0.0 Init/Exit
0.0 Synchronization
0.1 Barrier
0.0 Barrier Completion
9.5 Wait at Barrier
0.0 Overhead
100.0 Visits
0.0 Communication Matrix

0.0 <<time step loop>>
0.0 updatedt
0.1 updatex
8.7 updateien
0.0 gene
0.0 <<iteration loop>>
2.3 genbc
0.0 setd
0.0 newx
0.0 genf
0.0 genu
0.0 genh
0.0 newd
0.0 ewdtimerstamp
0.0 ewdmoreloc_
0.0 blkins3dst
0.0 blkrhs
0.0 ewdscatter2
0.0 ewdscatter2i
0.0 ewdmalloc_
0.0 MPI_Send
11.2 MPI_Recv
0.0 ewdfree_
0.0 ewdstatrhs
0.0 ewdbsrgetdiag
0.0 ewdmakesg
0.0 ewdmult
0.0 ewdgmres
5.9 ewdgather1
0.0 ewdbsrmatvec
0.0 ewdcopy
0.0 updated
0.2 recoverstress
1.0 checkd
0.0 ewdtimerstamp

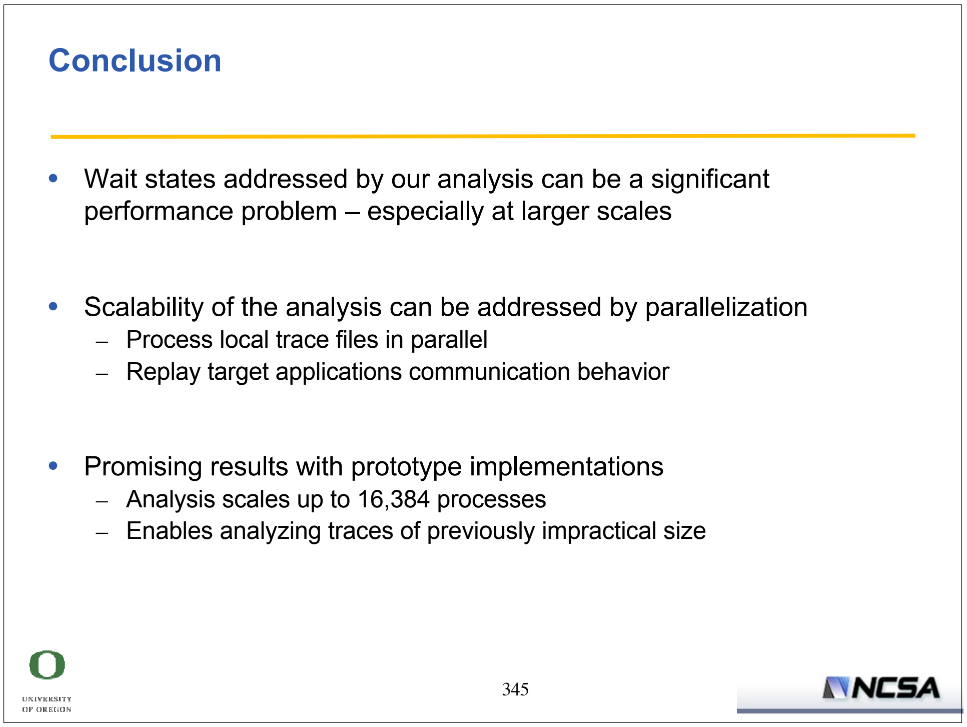
13,776,079 (29.4%) 4.688e+04
5,246,336 (11.2%) 4.688e+04
<1.281 ± 37.8%> 2.016e+00
4096 x 1

0.0 <<time step loop>>
0.0 updatedt
0.0 updatex
0.0 updateien
0.0 gene
0.0 <<iteration loop>>
0.0 genbc
0.0 setd
0.0 newx
0.0 genf
0.0 genu
0.0 genh
0.0 newd
0.0 ewdtimerstamp
0.0 ewdbarrier
9.5 MPI_Barrier
0.0 ewdmoreloc_
0.0 blkins3dst
0.0 blkrhs
0.0 ewdscatter2
0.0 ewdstatrhs
0.0 ewdbsrgetdiag
0.0 ewdmakesg
0.0 ewdmult
0.0 ewdgmres
0.0 ewdgather1
0.0 ewdbsrmatvec
0.0 ewdcopy
0.0 updated
0.0 recoverstress
0.0 checkd
0.0 ewdtimerstamp

4,455,566 (9.5%) 4.688e+04
<1.000 ± 36.3%> 2.782e+00

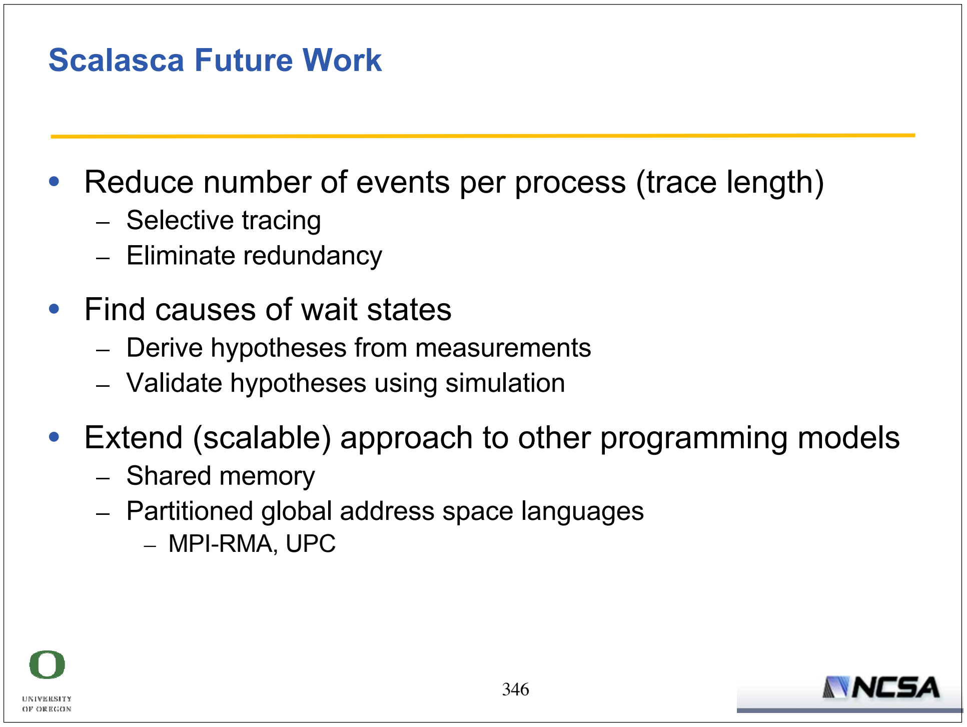
UNIVERSITY OF OREGON

NCSA



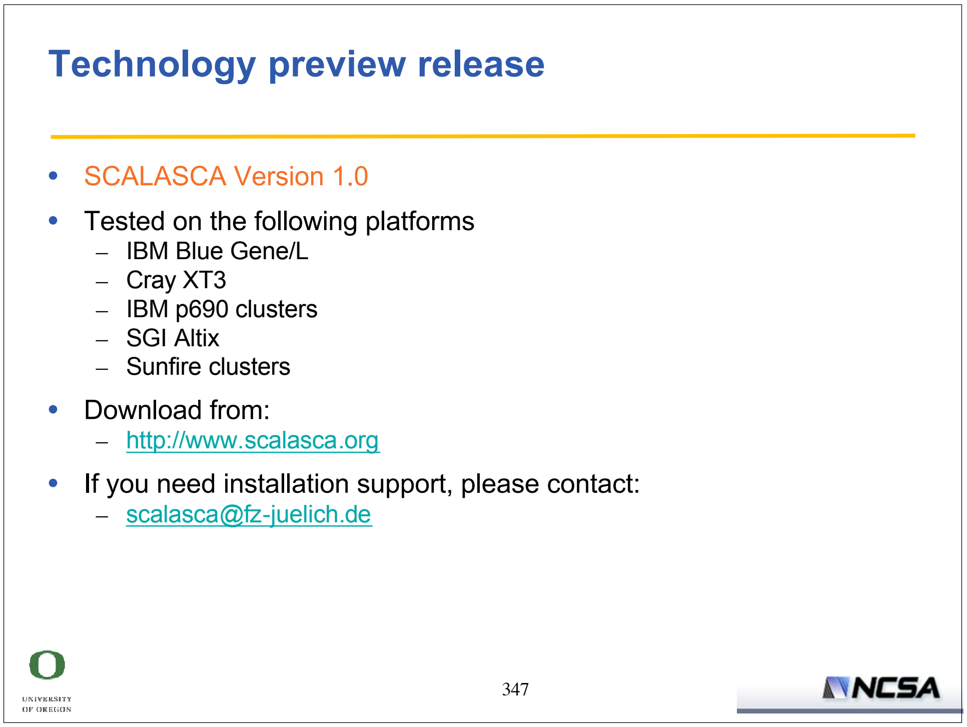
Conclusion

- Wait states addressed by our analysis can be a significant performance problem – especially at larger scales
- Scalability of the analysis can be addressed by parallelization
  - Process local trace files in parallel
  - Replay target applications communication behavior
- Promising results with prototype implementations
  - Analysis scales up to 16,384 processes
  - Enables analyzing traces of previously impractical size



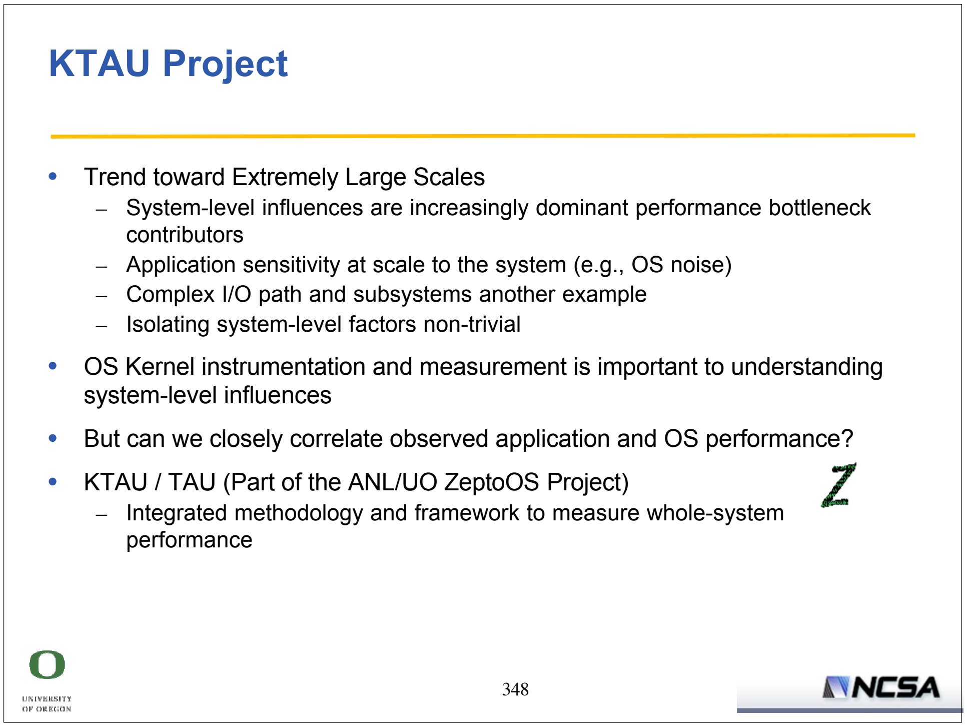
Scalasca Future Work

- Reduce number of events per process (trace length)
  - Selective tracing
  - Eliminate redundancy
- Find causes of wait states
  - Derive hypotheses from measurements
  - Validate hypotheses using simulation
- Extend (scalable) approach to other programming models
  - Shared memory
  - Partitioned global address space languages
    - MPI-RMA, UPC



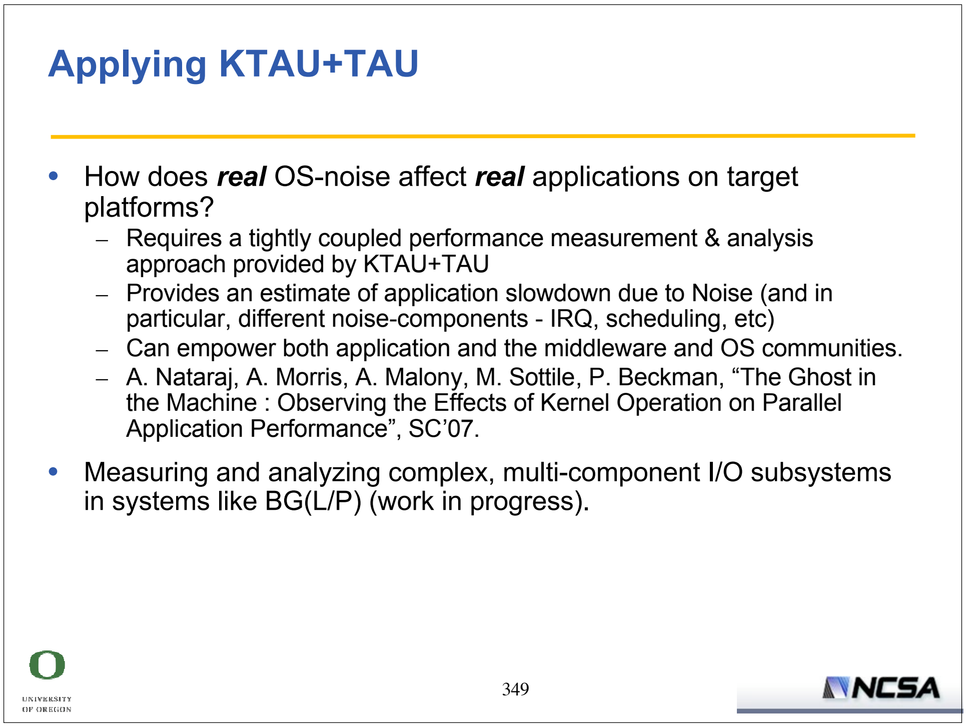
Technology preview release

- SCALASCA Version 1.0
- Tested on the following platforms
  - IBM Blue Gene/L
  - Cray XT3
  - IBM p690 clusters
  - SGI Altix
  - Sunfire clusters
- Download from:
  - http://www.scalasca.org
- If you need installation support, please contact:
  - scalasca@fz-juelich.de



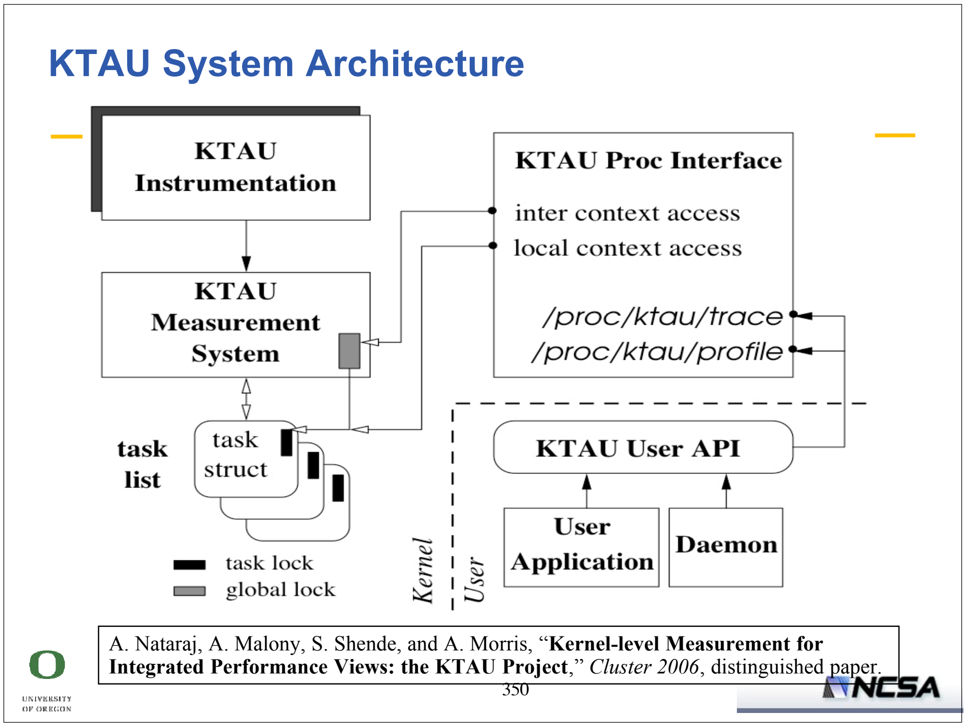
KTAU Project

- Trend toward Extremely Large Scales
  - System-level influences are increasingly dominant performance bottleneck contributors
  - Application sensitivity at scale to the system (e.g., OS noise)
  - Complex I/O path and subsystems another example
  - Isolating system-level factors non-trivial
- OS Kernel instrumentation and measurement is important to understanding system-level influences
- But can we closely correlate observed application and OS performance?
- KTAU / TAU (Part of the ANL/UO ZeptoOS Project)
  - Integrated methodology and framework to measure whole-system performance



Applying KTAU+TAU

- How does ***real*** OS-noise affect ***real*** applications on target platforms?
  - Requires a tightly coupled performance measurement & analysis approach provided by KTAU+TAU
  - Provides an estimate of application slowdown due to Noise (and in particular, different noise-components - IRQ, scheduling, etc)
  - Can empower both application and the middleware and OS communities.
  - A. Nataraj, A. Morris, A. Malony, M. Sottile, P. Beckman, “The Ghost in the Machine : Observing the Effects of Kernel Operation on Parallel Application Performance”, SC’07.
- Measuring and analyzing complex, multi-component I/O subsystems in systems like BG(L/P) (work in progress).



KTAU System Architecture

KTAU Instrumentation

KTAU Measurement System

task list

task struct

▬ task lock

▨ global lock

KTAU Proc Interface

inter context access

local context access

/proc/ktau/trace

/proc/ktau/profile

KTAU User API

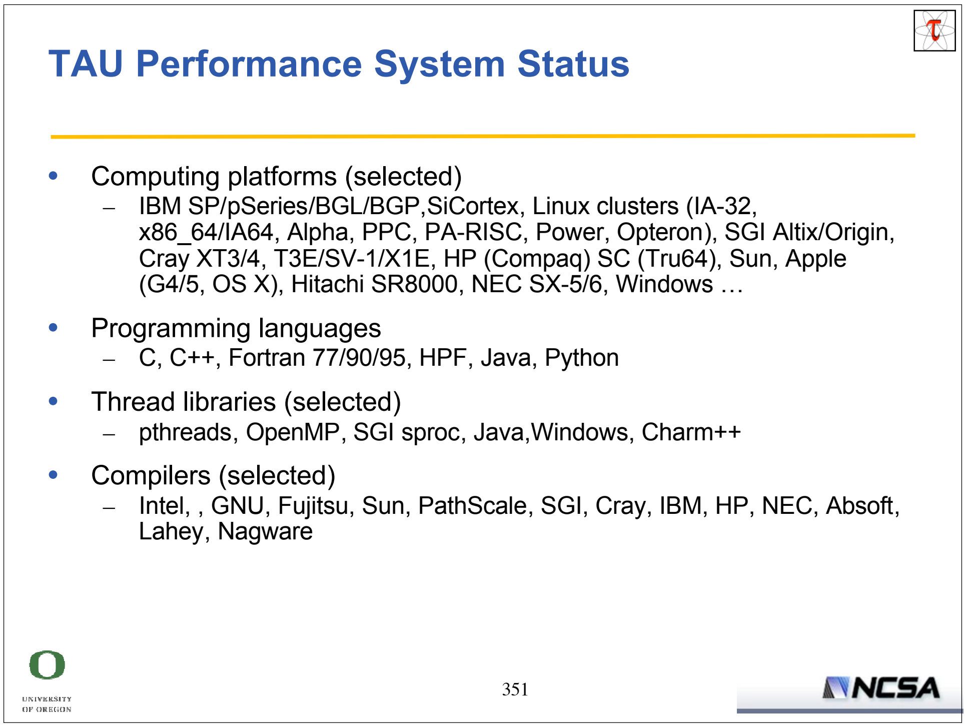
User Application

Daemon

Kernel

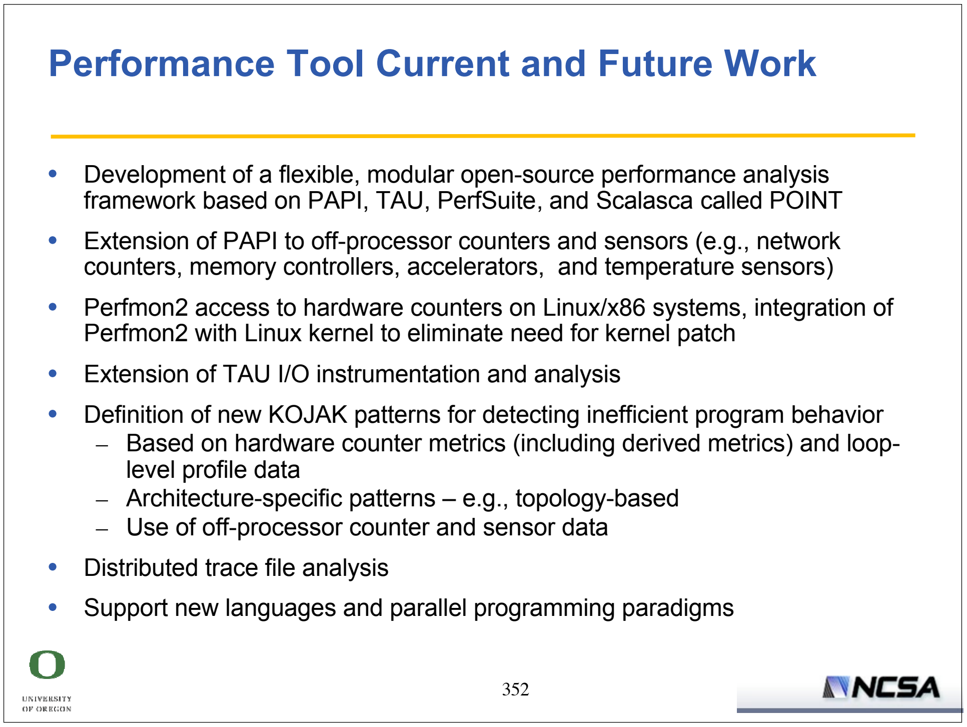
User

A. Nataraj, A. Malony, S. Shende, and A. Morris, “**Kernel-level Measurement for Integrated Performance Views: the KTAU Project**,” *Cluster 2006*, distinguished paper.



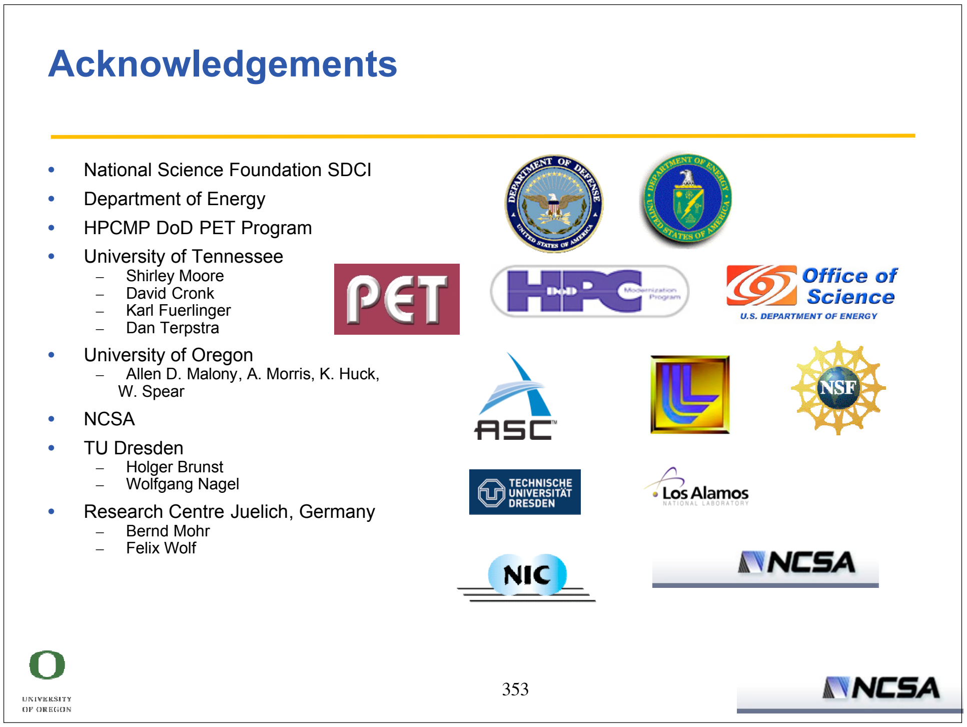
TAU Performance System Status

- Computing platforms (selected)
  - IBM SP/pSeries/BGL/BGP,SiCortex, Linux clusters (IA-32, x86_64/IA64, Alpha, PPC, PA-RISC, Power, Opteron), SGI Altix/Origin, Cray XT3/4, T3E/SV-1/X1E, HP (Compaq) SC (Tru64), Sun, Apple (G4/5, OS X), Hitachi SR8000, NEC SX-5/6, Windows …
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, Python
- Thread libraries (selected)
  - pthreads, OpenMP, SGI sproc, Java,Windows, Charm++
- Compilers (selected)
  - Intel, , GNU, Fujitsu, Sun, PathScale, SGI, Cray, IBM, HP, NEC, Absoft, Lahey, Nagware



Performance Tool Current and Future Work

- Development of a flexible, modular open-source performance analysis framework based on PAPI, TAU, PerfSuite, and Scalasca called POINT
- Extension of PAPI to off-processor counters and sensors (e.g., network counters, memory controllers, accelerators, and temperature sensors)
- Perfmon2 access to hardware counters on Linux/x86 systems, integration of Perfmon2 with Linux kernel to eliminate need for kernel patch
- Extension of TAU I/O instrumentation and analysis
- Definition of new KOJAK patterns for detecting inefficient program behavior
  - Based on hardware counter metrics (including derived metrics) and loop-level profile data
  - Architecture-specific patterns – e.g., topology-based
  - Use of off-processor counter and sensor data
- Distributed trace file analysis
- Support new languages and parallel programming paradigms



Acknowledgements

- National Science Foundation SDCI
- Department of Energy
- HPCMP DoD PET Program
- University of Tennessee
  - Shirley Moore
  - David Cronk
  - Karl Fuerlinger
  - Dan Terpstra
- University of Oregon
  - Allen D. Malony, A. Morris, K. Huck, W. Spear
- NCSA
- TU Dresden
  - Holger Brunst
  - Wolfgang Nagel
- Research Centre Juelich, Germany
  - Bernd Mohr
  - Felix Wolf