TAU Performance Tools

Sameer Shende, Allen D. Malony, and Alan Morris

{sameer, malony, amorris}@cs.uoregon.edu

Department of Computer and Information Science

NeuroInformatics Center

University of Oregon

UNIVERSITY
OF OREGON

Acknowledgements

- Pete Beckman, ANL
- Suravee Suthikulpanit, U. Oregon
- Aroon Nataraj, U. Oregon
- Katherine Riley, ANL

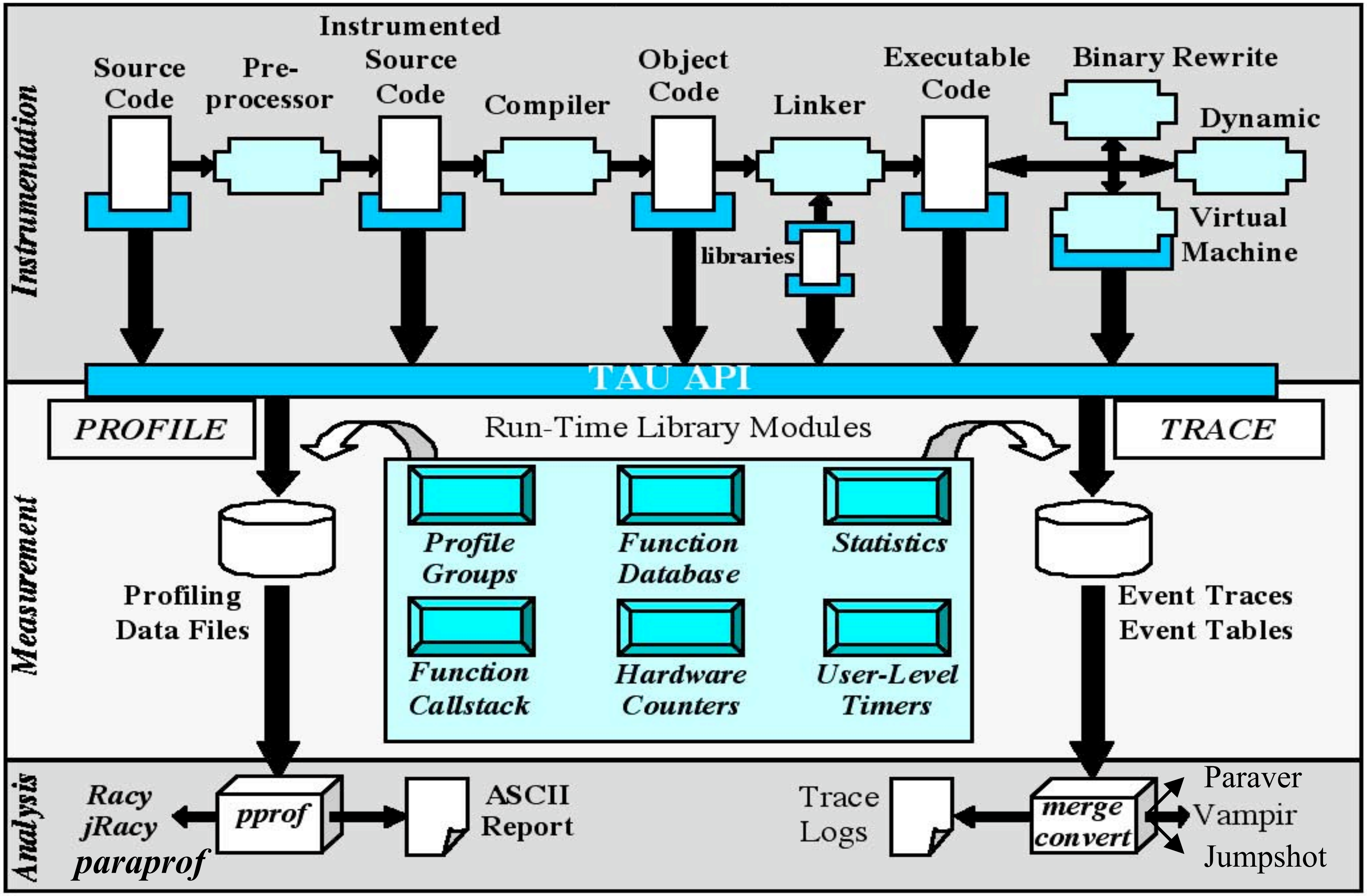
Outline

- Overview of features
- Instrumentation
- Measurement
- Analysis tools
- Linux kernel profiling with TAU

TAU Performance System Framework

- Tuning and Analysis Utilities
- Performance system framework for scalable parallel and distributed high-performance computing
- Targets a general complex system computation model
  - nodes / contexts / threads
  - Multi-level: system / software / parallelism
  - Measurement and analysis abstraction
- Integrated toolkit for performance instrumentation, measurement, analysis, and visualization
  - Portable, configurable performance profiling/tracing facility
  - Open software approach
- University of Oregon, LANL, FZJ Germany
- http://www.cs.uoregon.edu/research/paracomp/tau

TAU Performance System Architecture



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*)
  - Proxy Components

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 Measurement System Configuration

- configure [OPTIONS]
  - {-c++=<CC>, -cc=<cc>} — Specify C++ and C compilers
  - {-pthread, -sproc} — Use pthread or SGI sproc threads
  - -openmp — Use OpenMP threads
  - -jdk=<dir> — Specify Java instrumentation (JDK)
  - -opari=<dir> — Specify location of Opari OpenMP tool
  - -papi=<dir> — Specify location of PAPI
  - -pdt=<dir> — Specify location of PDT
  - -dyninst=<dir> — Specify location of DynInst Package
  - -mpi[inc/lib]=<dir> — Specify MPI library instrumentation
  - -shmem[inc/lib]=<dir> — Specify PSHMEM library instrumentation
  - -python[inc/lib]=<dir> — Specify Python instrumentation
  - -epilog=<dir> — Specify location of EPILOG
  - -vtf=<dir> — Specify location of VTF3 trace package
  - -arch=<architecture> (bgl,ibm64,ibm64linux…) — Specify architecture explicitly

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
  - -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 –arch=bgl –mpi –pdt=/usr/pdtoolkit-3.3.1 -pdt_c++=xlC
  - Use IBM BlueGene/L arch, XL compilers, MPI and PDT
  - Builds <tau>/bgl/bin/tau_instrumentor (executes on the front-end) and <tau>/bgl/lib/Makefile.tau-mpi-pdt stub
- ./configure –TRACE –PROFILE –arch=bgl –mpi
  - Enable both TAU profiling and tracing
- ./configure -c++=xlC_r -cc=xlc_r -mpi –pdt=/home/pdtoolkit-3.3.1 –TRACE –vtf=/usr/vtf3-1.33
  - Use IBM's xlC_r and xlc_r compilers with VTF3, PDT, MPI packages and multiple counters for measurements on the ppc64 front-end node
- Typically configure multiple measurement libraries

TAU Performance Framework Interfaces

- PDT [U. Oregon, LANL, FZJ] for instrumentation of C++, C99, F95 source code
- PAPI [UTK] & PCL[FZJ] 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/Intel® Trace Analyzer [Pallas/Intel]
- VTF3 trace generation library for Vampir [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]
- PerfDMF database can use Oracle, MySQL or PostgreSQL (IBM DB2 support planned)

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
  - 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

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 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()
    - Triggers one sample every 10 secs
  - call TAU_TRACK_MEMORY_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()
    - To turn off recording memory utilization
  - call TAU_ENABLE_TRACKING_MEMORY()
    - To re-enable tracking memory utilization

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

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 4^{th} invocation of foo.

Flat Profile – Pprof Profile Browser

- Intel Linux cluster
- F90 + MPICH
- Profile
  - Node
  - Context
  - Thread
- Events
  - code
  - MPI

emacs@neutron.cs.uoregon.edu

Buffers Files Tools Edit Search Mule Help

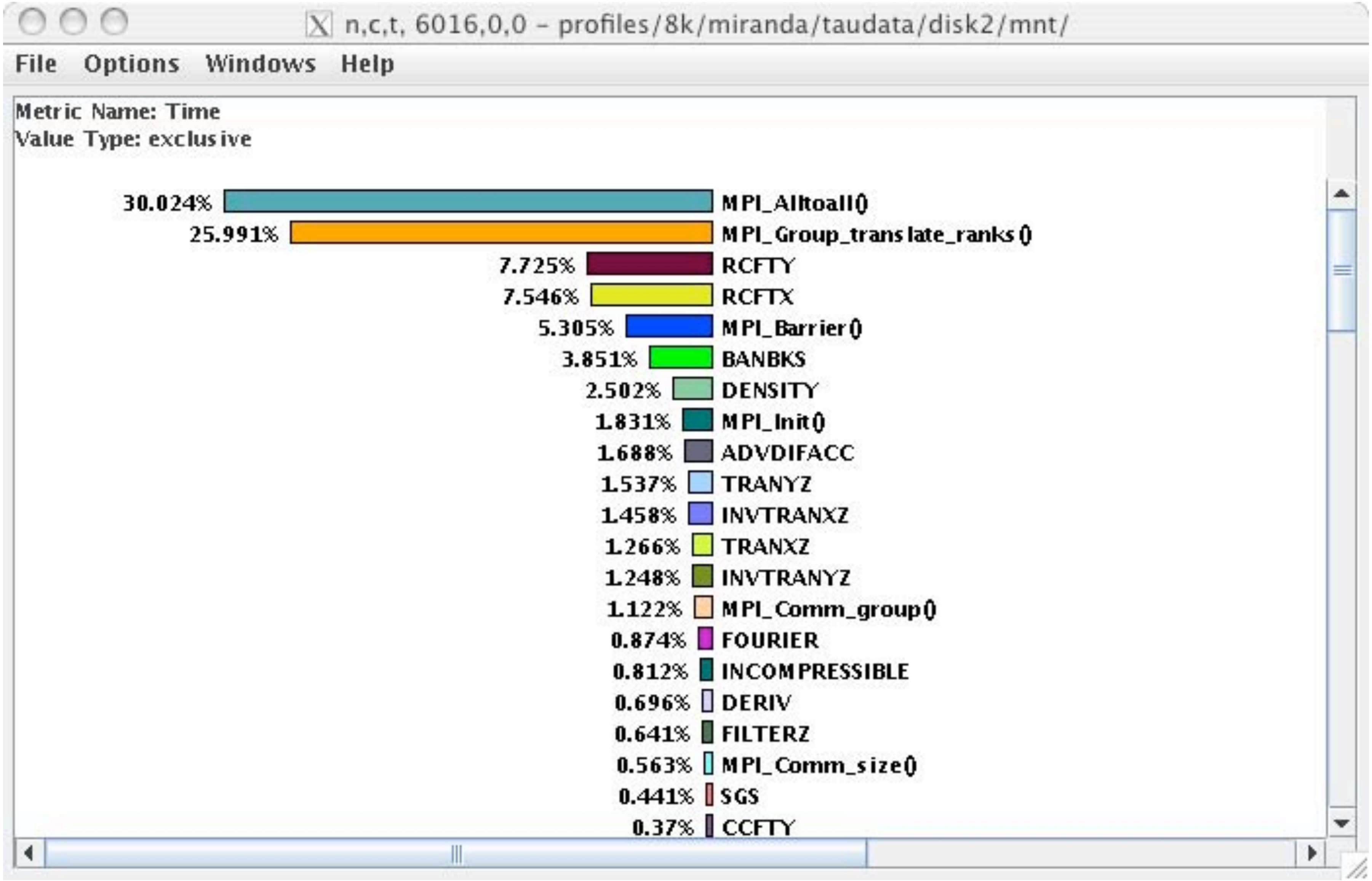
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:

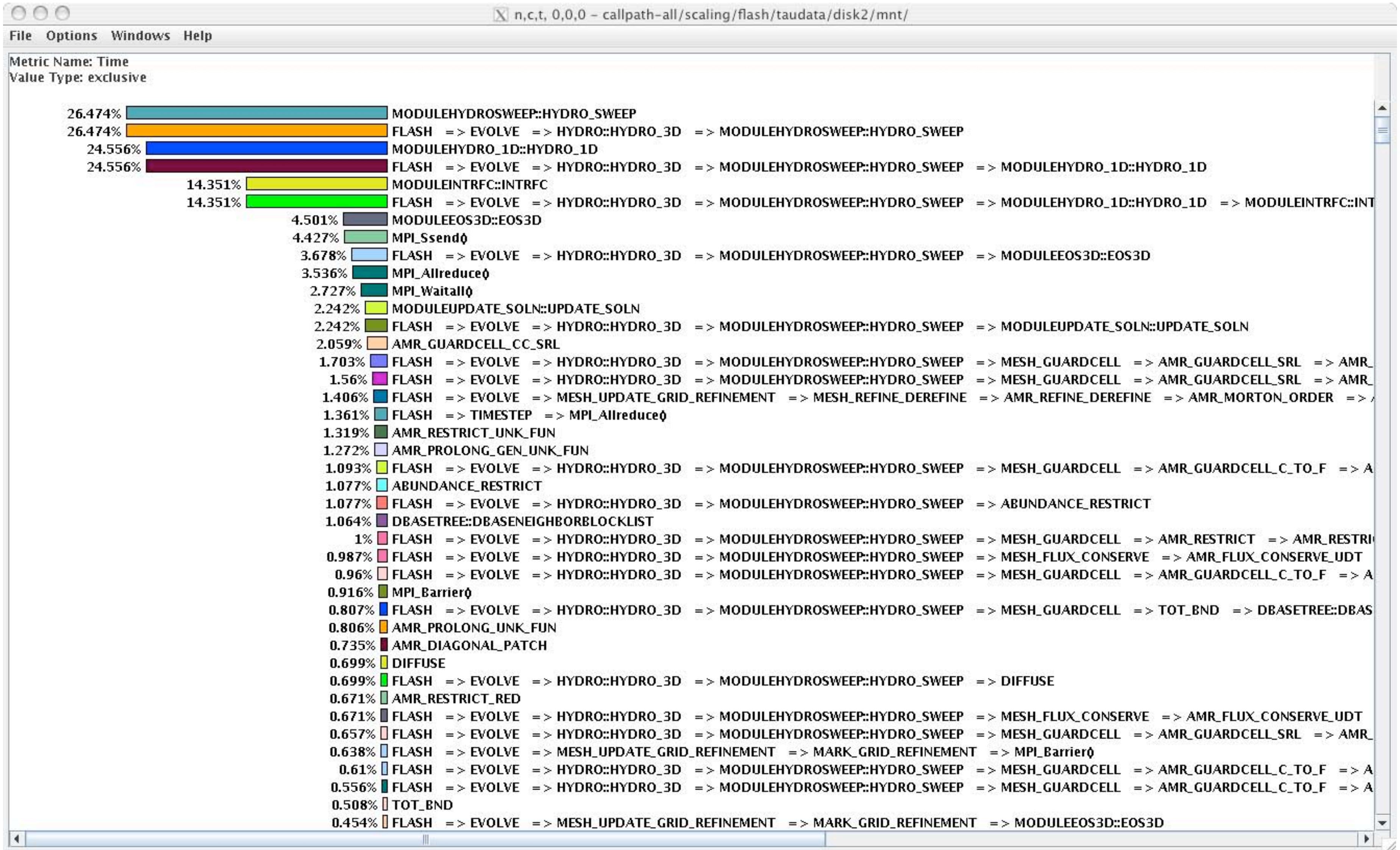
%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive usec/call	Name
100.0	1	3:11.293	1	15	191293269	applu
99.6	3,667	3:10.463	3	37517	63487925	bcast_inputs
67.1	491	2:08.326	37200	37200	3450	exchange_1
44.5	6,461	1:25.159	9300	18600	9157	buts
41.0	1:18.436	1:18.436	18600	0	4217	MPI_Recv()
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--

Flat Profile – TAU's Paraprof Profile Browser



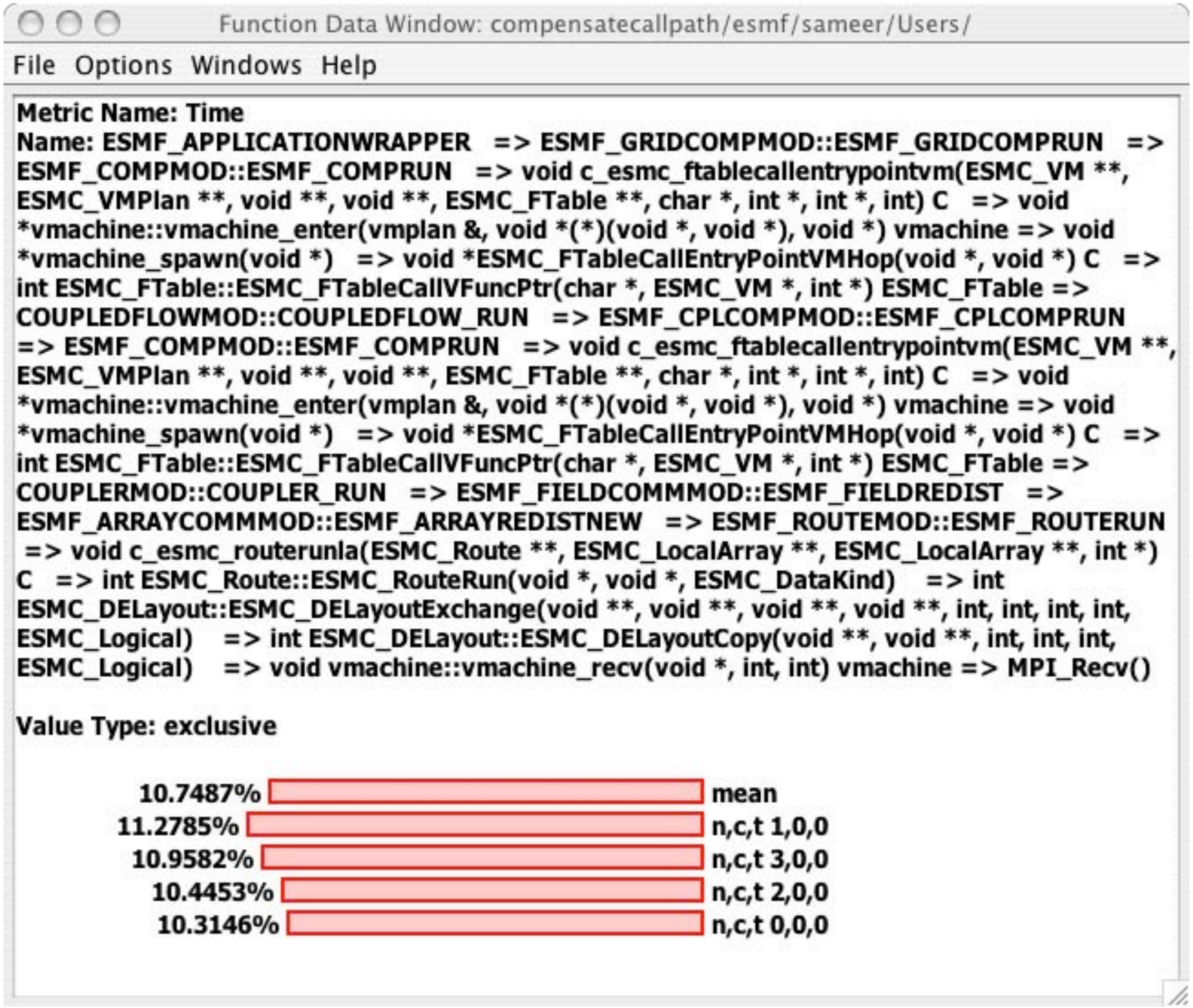
Callpath Profile



Callpath Profile - parent/node/child view

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

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

     1.8584        1.8584        1196/13188                     TOKEN_MODULE::TOKEN_GS_I   [521]
     0.584         0.584         234/13188                      TOKEN_MODULE::TOKEN_GS_L   [544]
     25.0819       25.0819       11758/13188                    TOKEN_MODULE::TOKEN_GS_R8   [734]
 --> 27.5242       27.5242       13188                          MPI_Waitall()   [525]


     17.9579       39.1657       156/156                        DERIVATIVE_MODULE::DERIVATIVES_NOFACE   [841]
 --> 17.9579       39.1657       156                            DERIVATIVE_MODULE::DERIVATIVES_FACE   [843]
     0.0156        0.0195        312/312                        TIMER_MODULE::TIMERSET   [77]
     0.1133        9.1269        2340/2340                      MESSAGE_MODULE::CLONE_GET_R8   [808]
     0.1602        11.4608       4056/4056                      MESSAGE_MODULE::CLONE_PUT_R8   [850]
     0.0059        0.6006        117/117                        MESSAGE_MODULE::CLONE_PUT_I   [856]


     14.1151       21.6209       5/5                            MATRIX_MODULE::MCGDS   [1443]
 --> 14.1151       21.6209       5                              MATRIX_MODULE::CSR_CG_SOLVER   [1470]
     0.0654        1.2617        1005/1005                      TOKEN_MODULE::TOKEN_GET_R8   [769]
     0.0557        5.2714        1005/1005                      TOKEN_MODULE::TOKEN_REDUCTION_R8_S   [1475]
     0.0703        0.9726        1000/1000                      TOKEN_MODULE::TOKEN_REDUCTION_R8_V   [208]
```

Callpath Profiling



Phase Profile – Dynamic Phases

In 51st iteration, time spent in MPI_Waitall was 85.81 secs

Total time spent in MPI_Waitall was 4137.9 secs across all 92 iterations

Call Path Data n,c,t, 0,0,0 – phase/mfix/taudata/disk2/mnt/

File Options Windows Help

Metric Name: Time
Sorted By: exclusive
Units: seconds

47.378	47.378	15712.0/1345134.0	ITERATE	47[6980]
65.217	65.217	21232.0/1345134.0	ITERATE	48[7116]
55.321	55.321	17888.0/1345134.0	ITERATE	49[7252]
51.351	51.351	16592.0/1345134.0	ITERATE	50[7388]
85.81	85.81	28208.0/1345134.0	ITERATE	51[7524]
75.069	75.069	24384.0/1345134.0	ITERATE	52[7670]
78.938	78.938	25728.0/1345134.0	ITERATE	53[7806]
69.684	69.684	23104.0/1345134.0	ITERATE	54[7942]
58.461	58.461	19072.0/1345134.0	ITERATE	55[8080]
85.117	85.117	27856.0/1345134.0	ITERATE	56[8216]
47.885	47.885	15504.0/1345134.0	ITERATE	57[8354]
46.436	46.436	14816.0/1345134.0	ITERATE	58[8490]
46.242	46.242	14752.0/1345134.0	ITERATE	59[8636]
45.728	45.728	14640.0/1345134.0	ITERATE	60[8772]
45.244	45.244	14656.0/1345134.0	ITERATE	61[8908]
45.283	45.283	14416.0/1345134.0	ITERATE	62[9044]
61.168	61.168	20032.0/1345134.0	ITERATE	63[9180]
46.992	46.992	15600.0/1345134.0	ITERATE	64[9326]
47.01	47.01	15792.0/1345134.0	ITERATE	65[9462]
44.046	44.046	14608.0/1345134.0	ITERATE	66[9598]
47.424	47.424	15584.0/1345134.0	ITERATE	67[9734]
41.176	41.176	13472.0/1345134.0	ITERATE	68[9870]
51.488	51.488	16880.0/1345134.0	ITERATE	69[10016]
43.714	43.714	14480.0/1345134.0	ITERATE	70[10152]
46.175	46.175	15152.0/1345134.0	ITERATE	71[10288]
45.348	45.348	14864.0/1345134.0	ITERATE	72[10424]
38.728	38.728	12848.0/1345134.0	ITERATE	73[10560]
46	46	15008.0/1345134.0	ITERATE	74[10706]
52.453	52.453	17008.0/1345134.0	ITERATE	75[10842]
44.341	44.341	14496.0/1345134.0	ITERATE	76[10978]
44.288	44.288	14240.0/1345134.0	ITERATE	77[11116]
58.298	58.298	18736.0/1345134.0	ITERATE	78[11252]
48.099	48.099	15584.0/1345134.0	ITERATE	79[11388]
45.351	45.351	14480.0/1345134.0	ITERATE	80[11534]
48.512	48.512	15824.0/1345134.0	ITERATE	81[11670]
41.185	41.185	13408.0/1345134.0	ITERATE	82[11806]
34.789	34.789	11248.0/1345134.0	ITERATE	83[11944]
34.061	34.061	10944.0/1345134.0	ITERATE	84[12080]
33.843	33.843	10960.0/1345134.0	ITERATE	85[12216]
33.182	33.182	10848.0/1345134.0	ITERATE	86[12362]
33.165	33.165	10752.0/1345134.0	ITERATE	87[12498]
29.992	29.992	9632.0/1345134.0	ITERATE	88[12634]
28.337	28.337	9136.0/1345134.0	ITERATE	89[12770]
35.926	35.926	11488.0/1345134.0	ITERATE	90[12906]
36.238	36.238	11648.0/1345134.0	ITERATE	91[13052]
27.385	27.385	8896.0/1345134.0	ITERATE	92[13188]
--> 4137.9	4137.9	1345134.0	MPI_Waitall()[121]	

Using TAU

- Install TAU

  % configure ; make clean install
- Instrument application
  - TAU Profiling API
- Typically modify application makefile
  - include TAU's stub makefile, modify variables
- Set environment variables
  - directory where profiles/traces are to be stored
  - name of merged trace file, retain intermediate trace files, etc.
- Execute application

  % mpirun –np <procs> a.out;
- Analyze performance data
  - paraprof, vampir/traceanalyzer, pprof, paraver …

AutoInstrumentation using TAU_COMPILER

- $(TAU_COMPILER) stub Makefile variable in 2.14+ release
- 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
- Uses original compilation command if an error occurs

TAU_COMPILER – Improving Integration in Makefiles

OLD

```
include /usr/tau-2.14/include/Makefile
CXX = mpCC
F90 = mpxlf90_r
PDTPARSE = $(PDTDIR)/
       $(PDTARCHDIR)/bin/cxxparse
TAUINSTR =
$(TAUROOT)/$(CONFIG_ARCH)/
       bin/tau_instrumentor
CFLAGS = $(TAU_DEFS) $(TAU_INCLUDE)
LIBS = $(TAU_MPI_LIBS) $(TAU_LIBS) -lm
OBJS = f1.o f2.o f3.o … fn.o

app: $(OBJS)
     $(CXX) $(LDFLAGS) $(OBJS) -o $@
       $(LIBS)
.cpp.o:
     $(PDTPARSE) $<
     $(TAUINSTR) $*.pdb $< -o
       $*.i.cpp -f select.dat
     $(CC) $(CFLAGS) -c $*.i.cpp
```

NEW

```
include /usr/tau-2.14/include/Makefile
CXX = $(TAU_COMPILER) mpCC
F90 = $(TAU_COMPILER) 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 $<
```

TAU_COMPILER Options

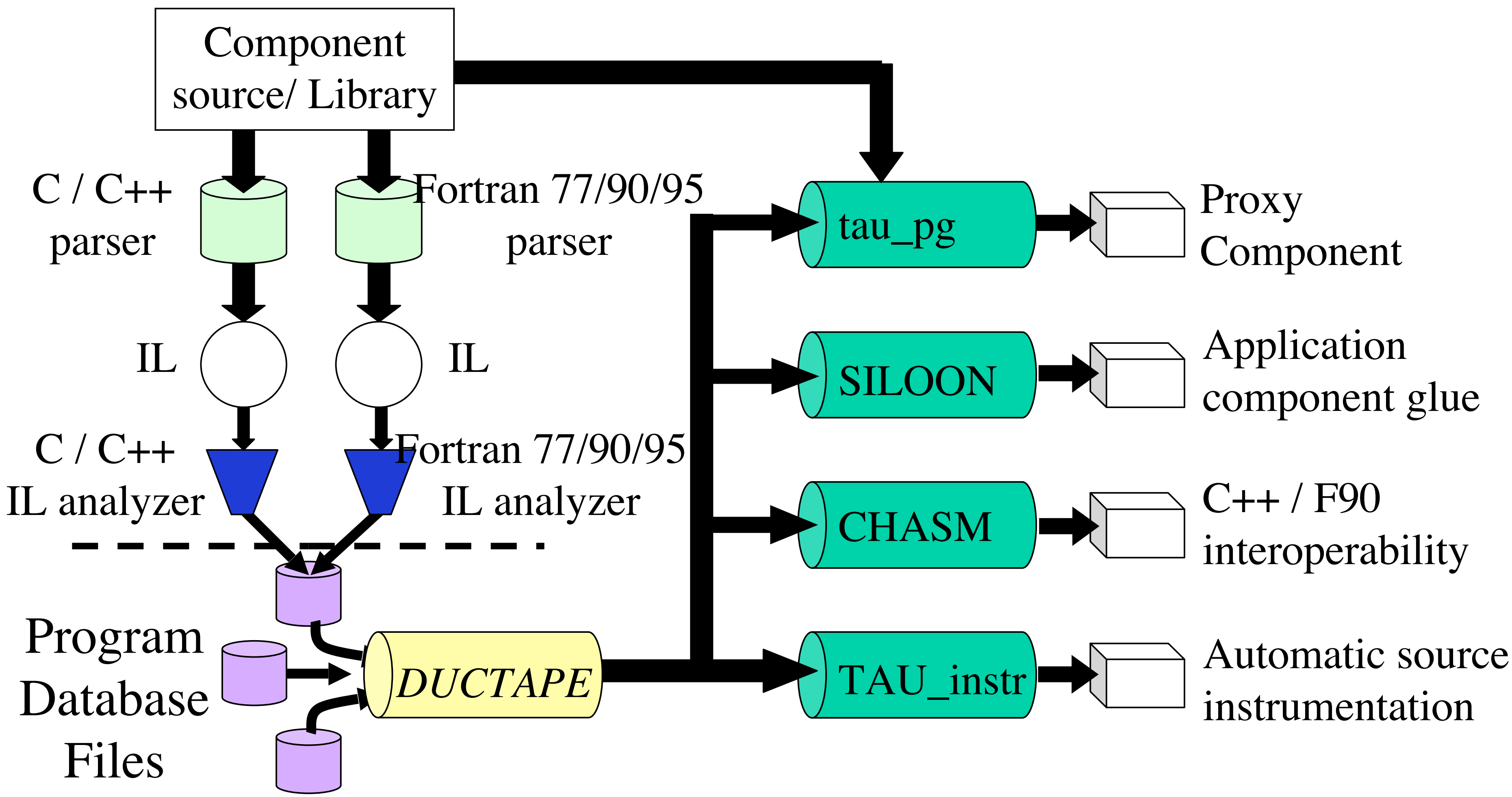
- Optional parameters for $(TAU_COMPILER):
  - -optVerbose Turn on verbose debugging messages
  - -optPdtDir="" PDT architecture directory. Typically $(PDTDIR)/$(PDTARCHDIR)
  - -optPdtF95Opts="" Options for Fortran parser in PDT (f95parse)
  - -optPdtCOpts="" Options for C parser in PDT (cparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - -optPdtCxxOpts="" Options for C++ parser in PDT (cxxparse). Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - -optPdtF90Parser="" Specify a different Fortran parser. For e.g., f90parse instead of f95parse
  - -optPdtUser="" Optional arguments for parsing source code
  - -optPDBFile="" Specify [merged] PDB file. Skips parsing phase.
  - -optTauInstr="" Specify location of tau_instrumentor. Typically $(TAUROOT)/$(CONFIG_ARCH)/bin/tau_instrumentor
  - -optTauSelectFile="" Specify selective instrumentation file for tau_instrumentor
  - -optTau="" Specify options for tau_instrumentor
  - -optCompile="" Options passed to the compiler. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE) $(TAU_DEFS)
  - -optLinking="" Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
  - -optNoMpi Removes -l*mpi* libraries during linking (default)
  - -optKeepFiles Does not remove intermediate .pdb and .inst.* files

```
e.g.,
OPT=-optTauSelectFile=select.tau -optPDBFile=merged.pdb
F90 = $(TAU_COMPILER) $(OPT) blrts_xlf90
```

Program Database Toolkit (PDT)

- Program code analysis framework
  - develop source-based tools
- *High-level interface* to source code information
- *Integrated toolkit* for source code parsing, database creation, and database query
  - Commercial grade front-end parsers
  - Portable IL analyzer, database format, and access API
  - Open software approach for tool development
- Multiple source languages
- Implement automatic performance instrumentation tools
  - *tau_instrumentor*

Program Database Toolkit



TAU Tracing Enhancements

- Configure TAU with -TRACE –vtf=dir option

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

  Generates tau_merge, tau2vtf tools in <tau>/ppc64/bin dir

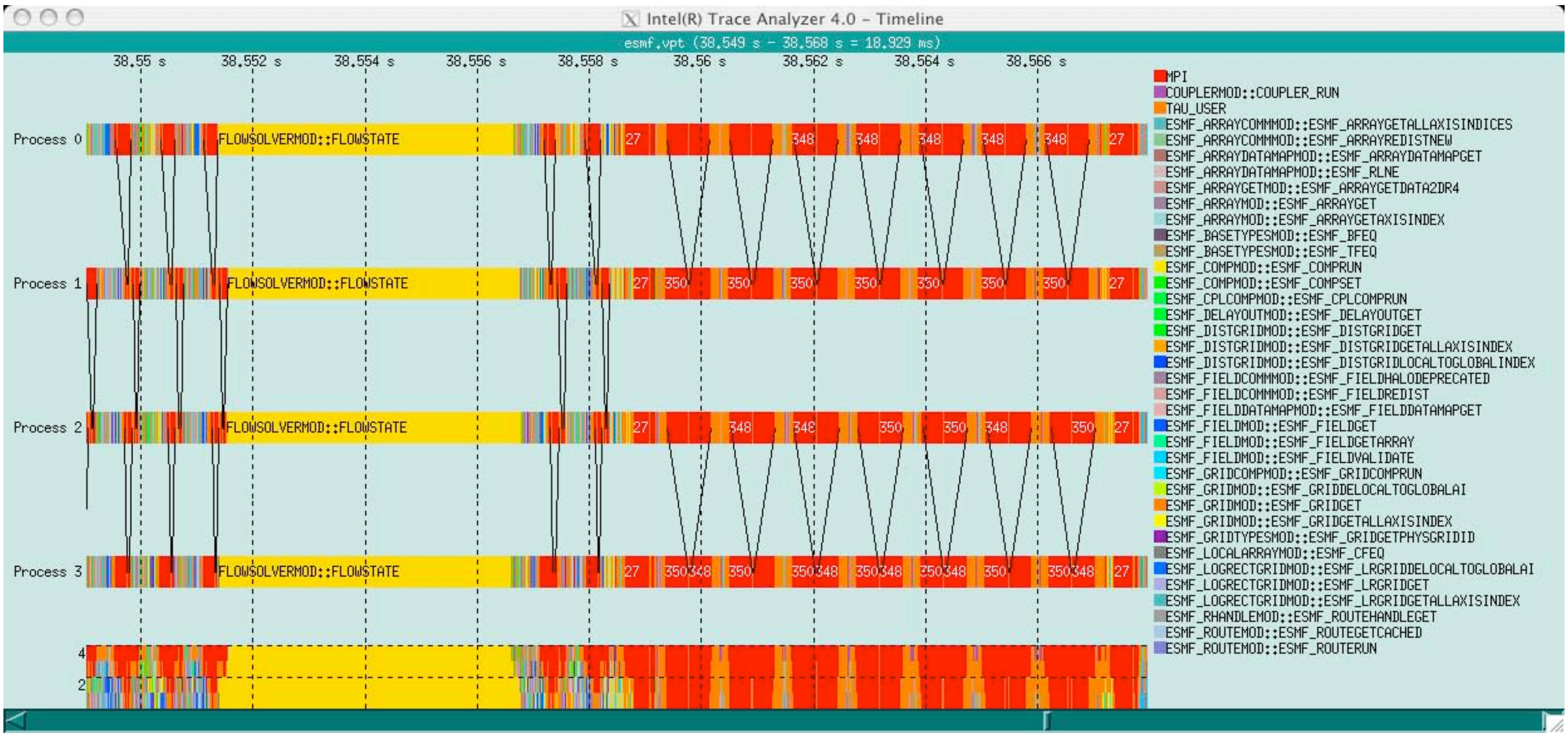
  ```
  % configure -arch=bgl -TRACE -pdt=<dir>
    -pdt_c++=xlC -mpi
  ```

  Generates library in <tau>/bgl/lib directory
- Execute application

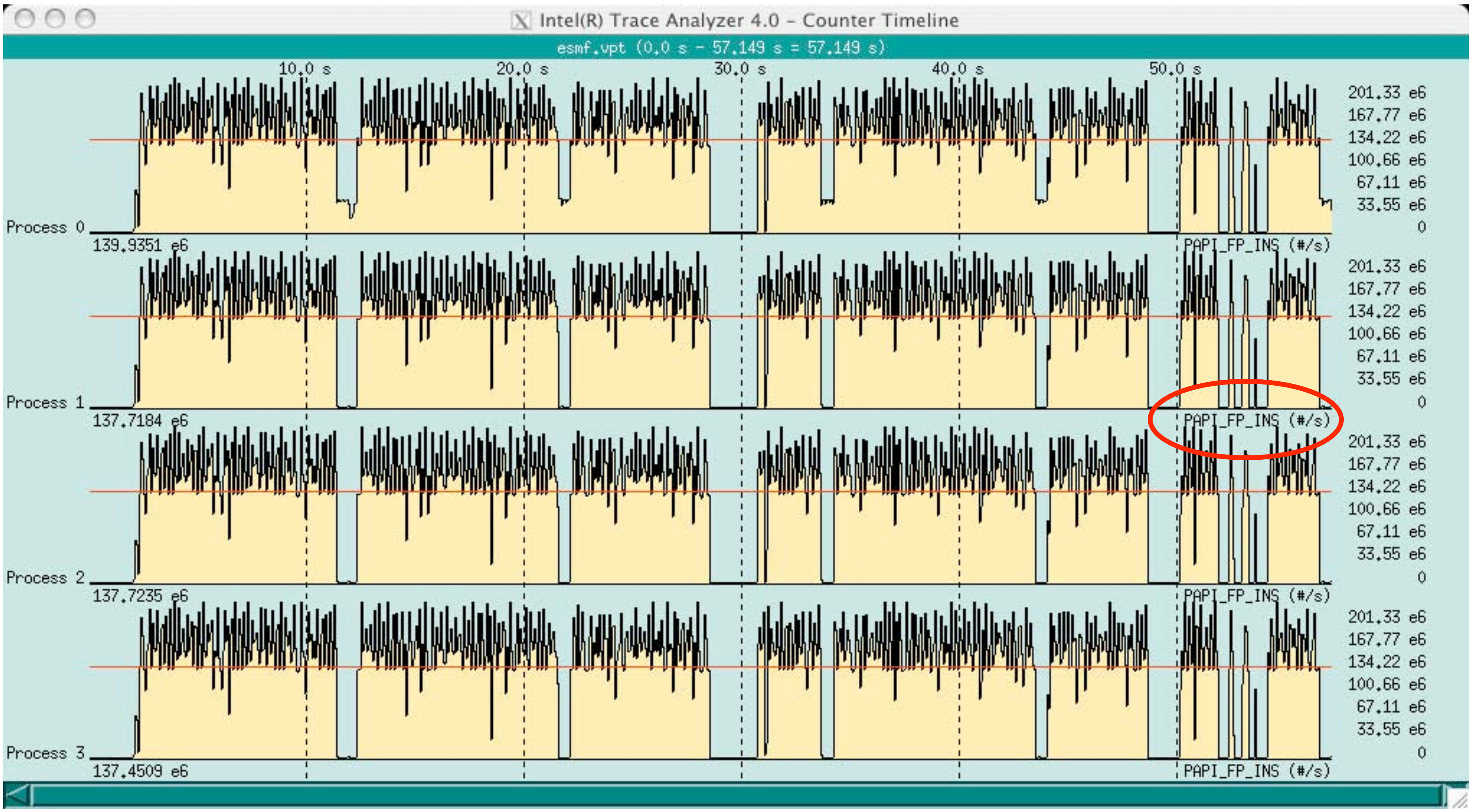
  ```
  % mpirun -partition Pgeneral2 -np 16 -cwd `pwd`
  -exe `pwd`/<app>
  ```
- Merge and convert trace files to VTF3 format

  ```
  % tau_merge *.trc app.trc
  % tau2vtf app.trc tau.edf app.vpt.gz
  % traceanalyzer foo.vpt.gz
  ```

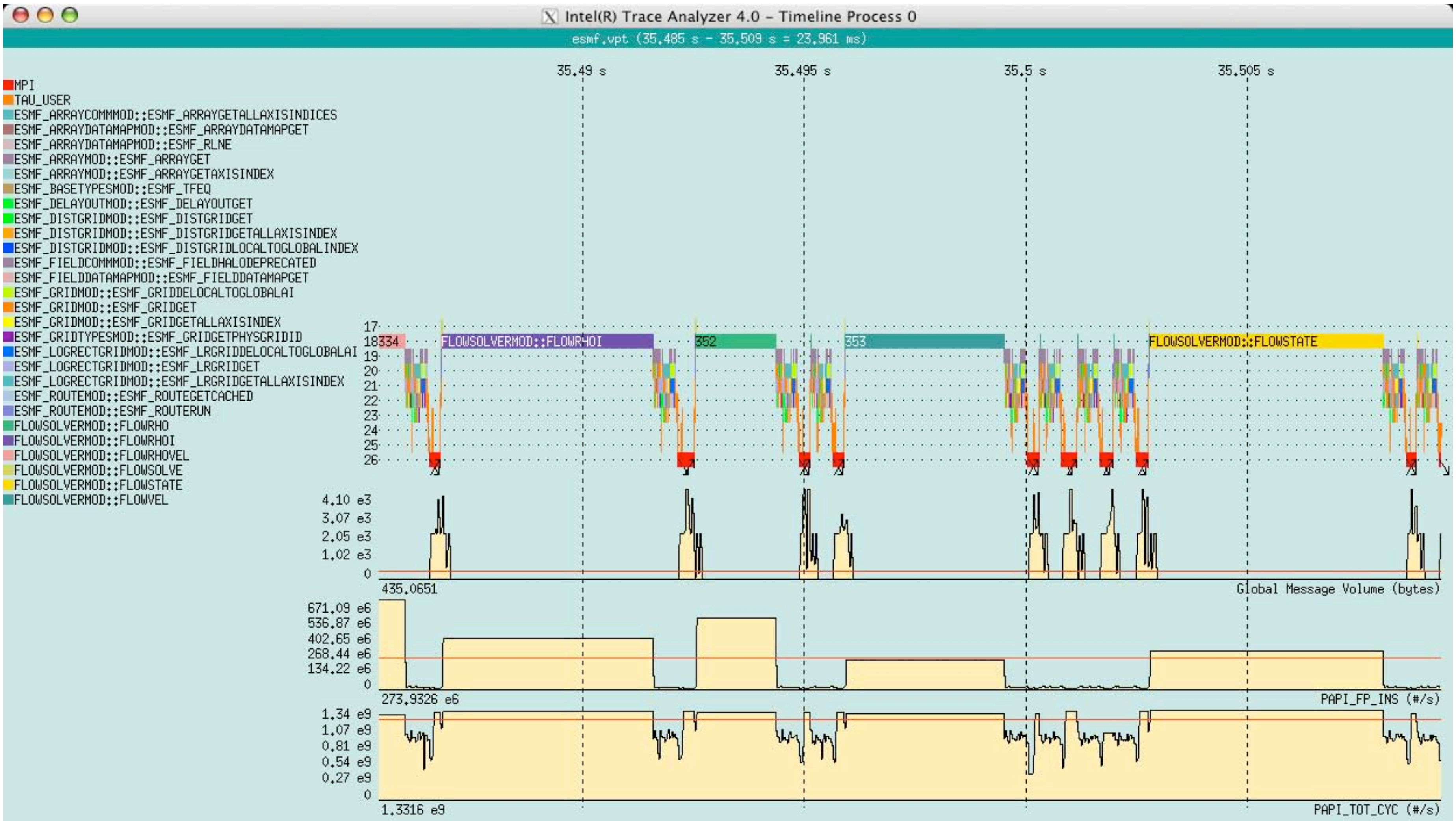
Intel ® Traceanalyzer (Vampir) Global Timeline



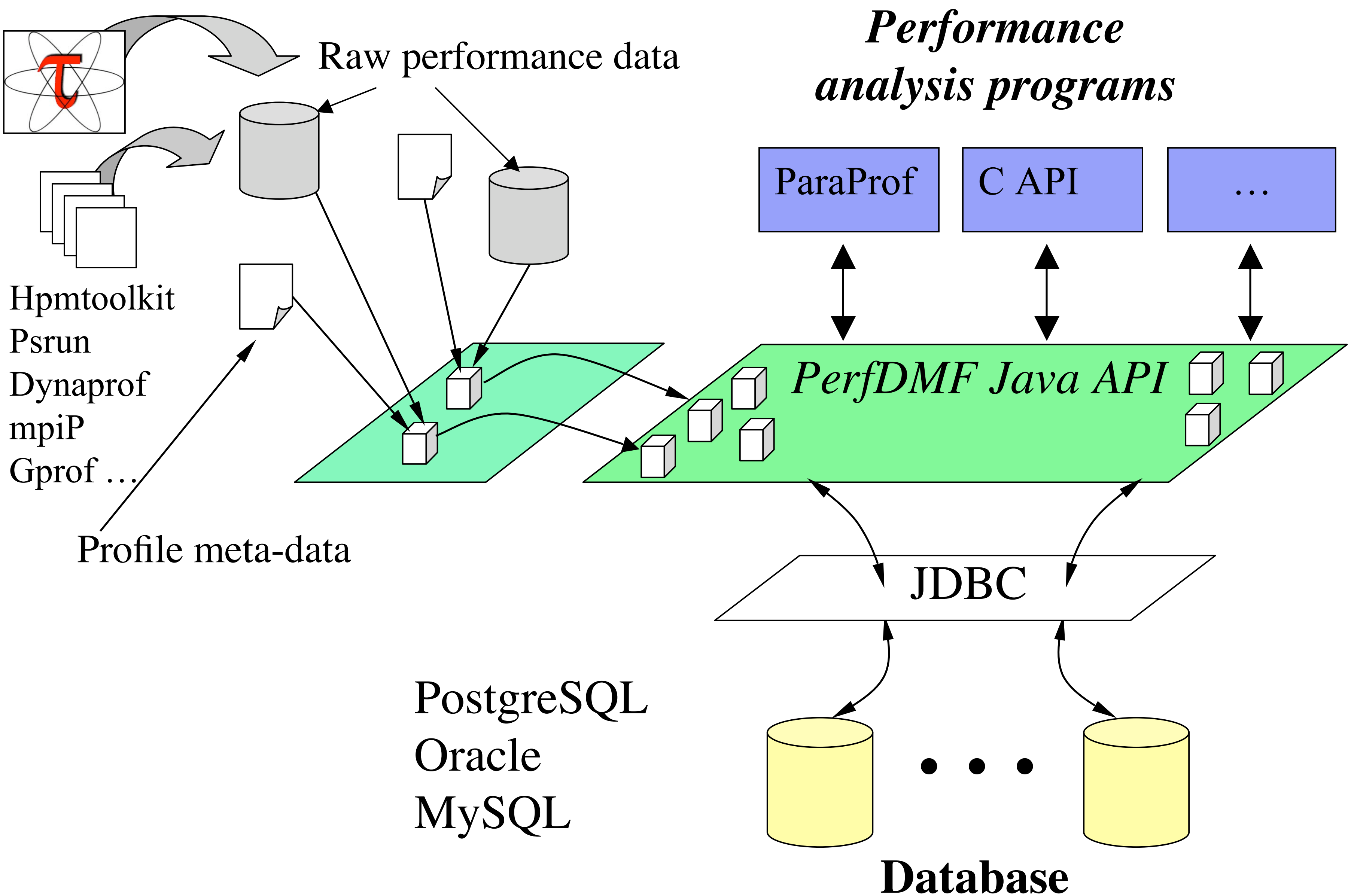
Visualizing TAU Traces with Counters/Samples



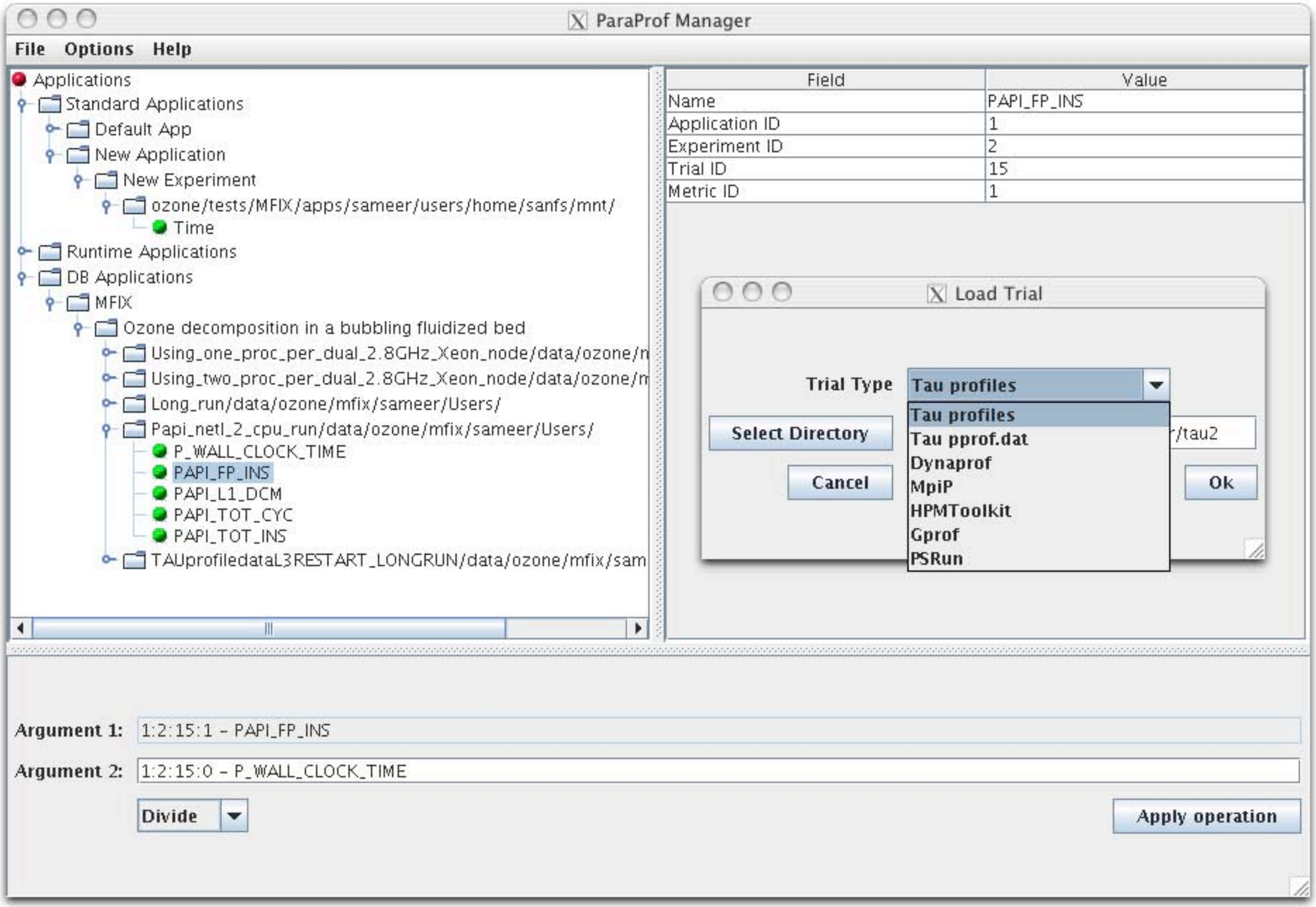
Visualizing TAU Traces with Counters/Samples



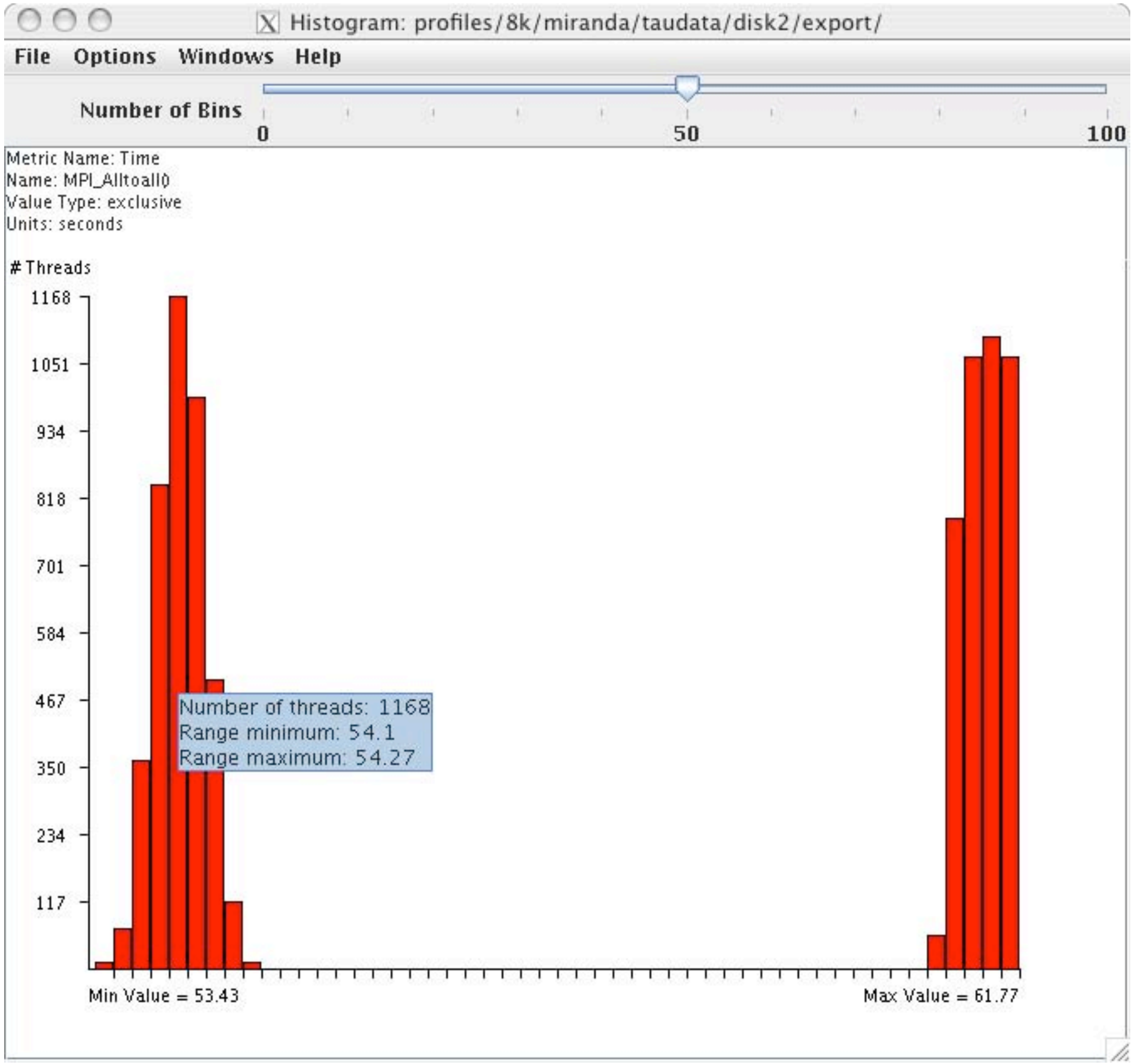
TAU Performance Data Management Framework



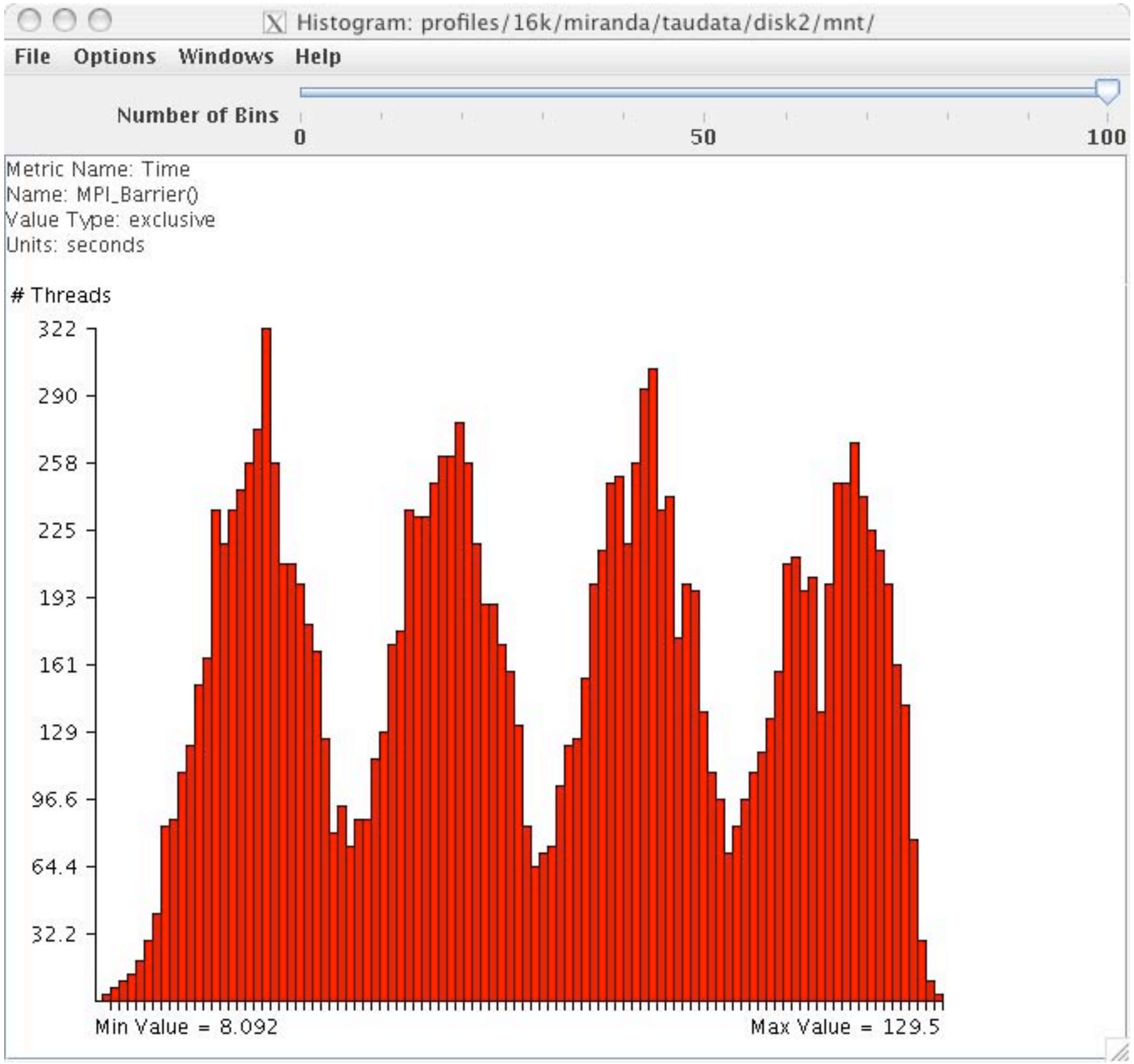
Paraprof Manager – Performance Database



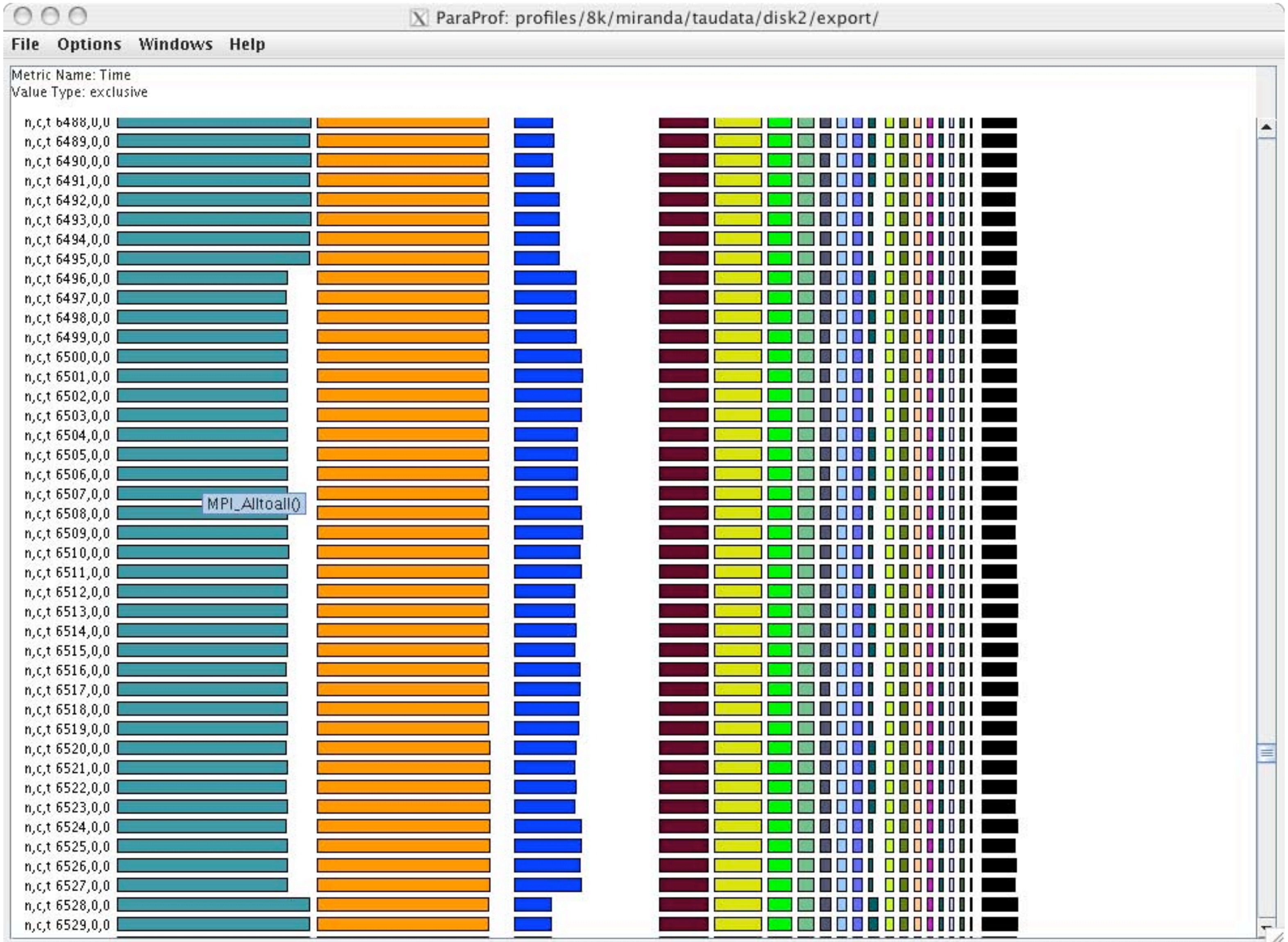
Paraprof Scalable Histogram View



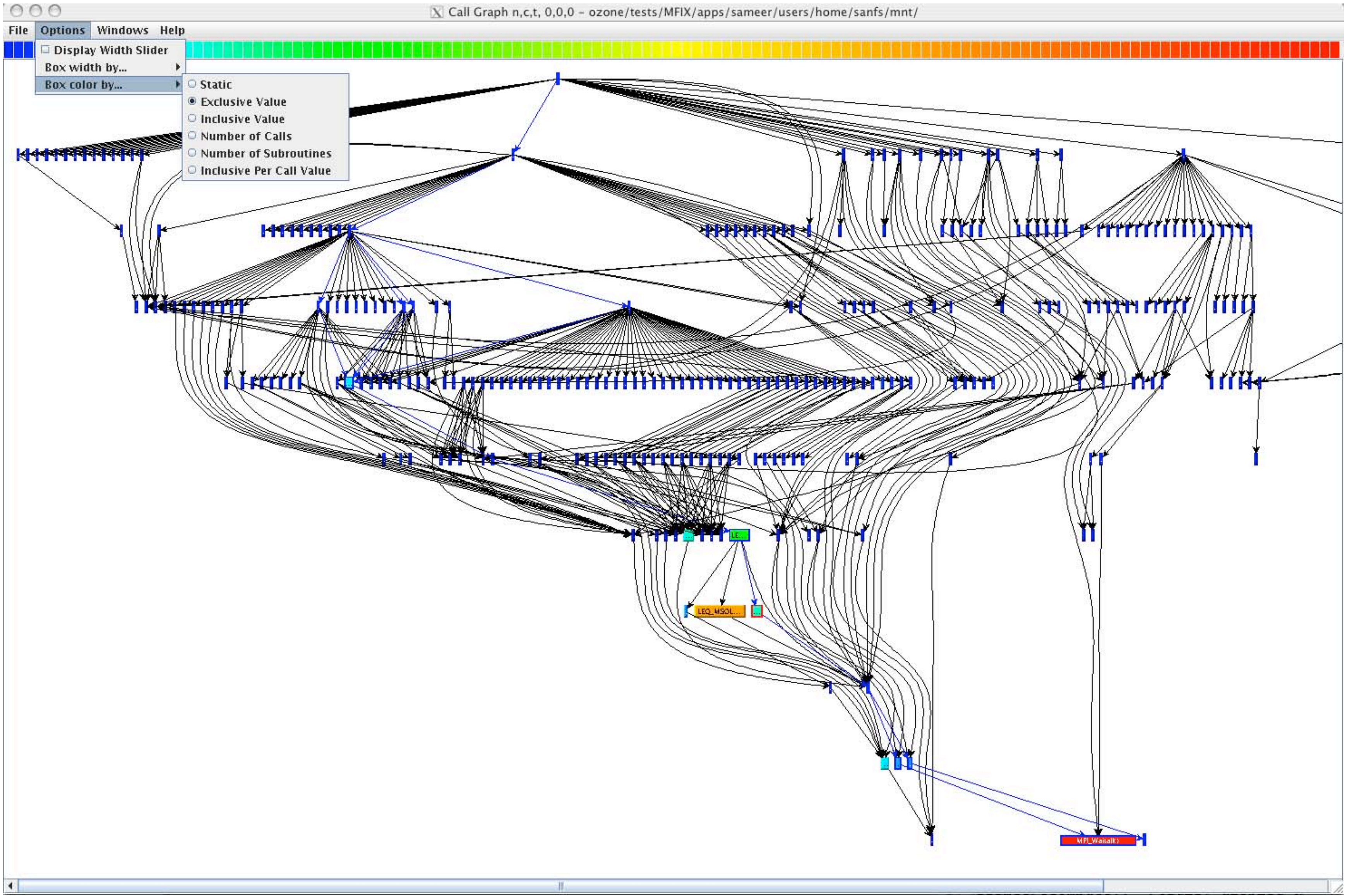
MPI_Barrier Histogram over 16K cpus of BG/L



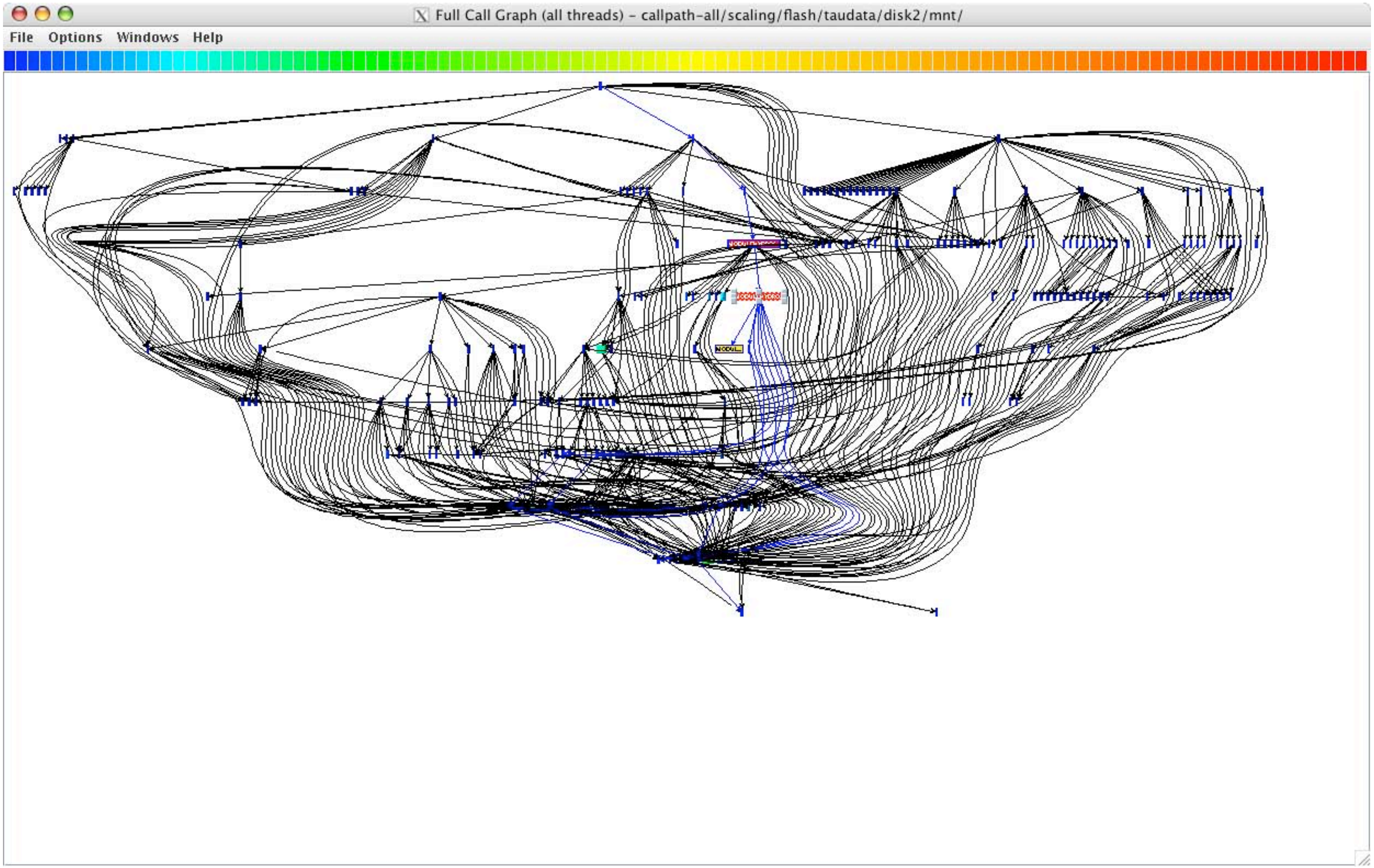
Paraprof Profile Browser



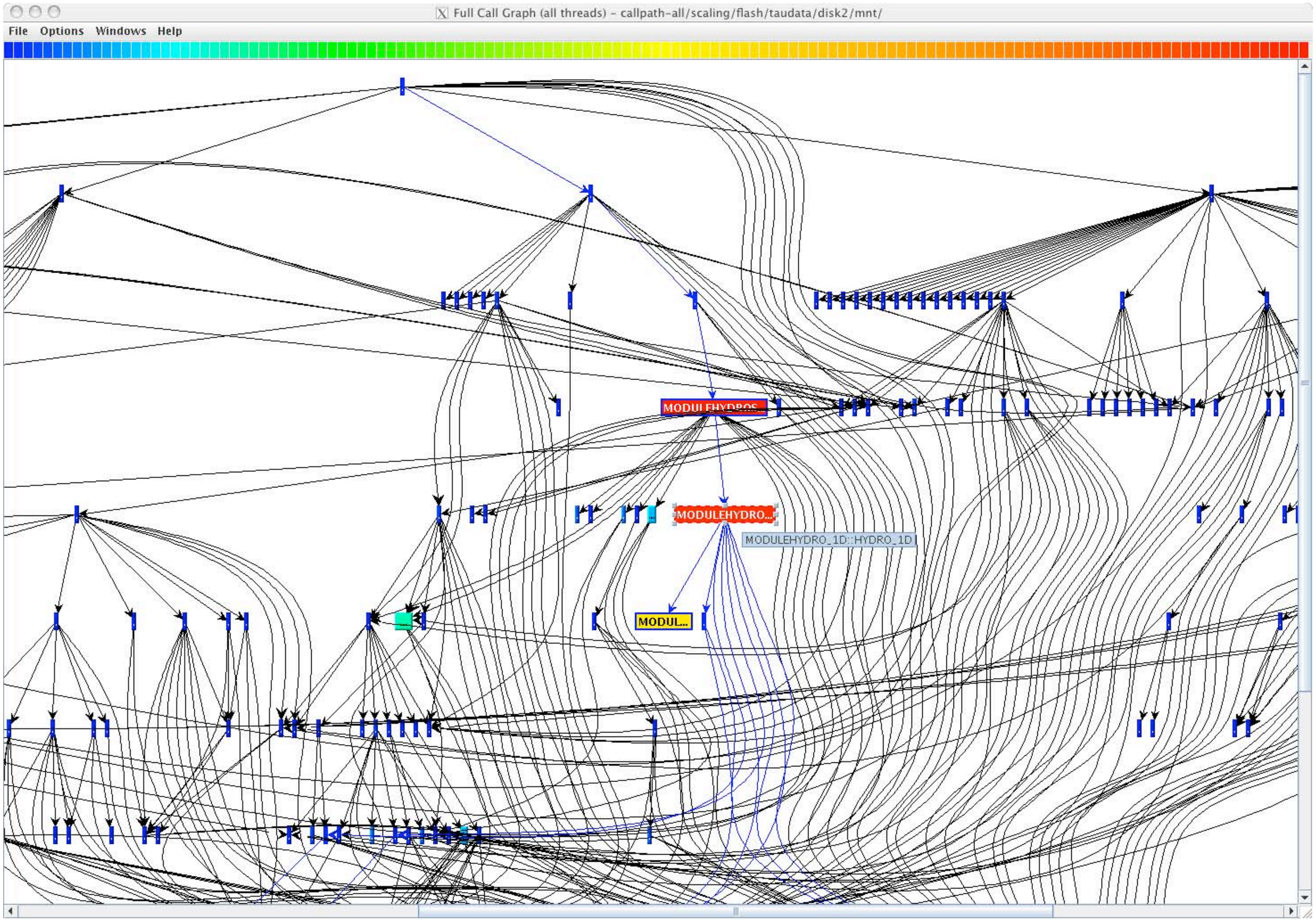
Paraprof – Full Callgraph View



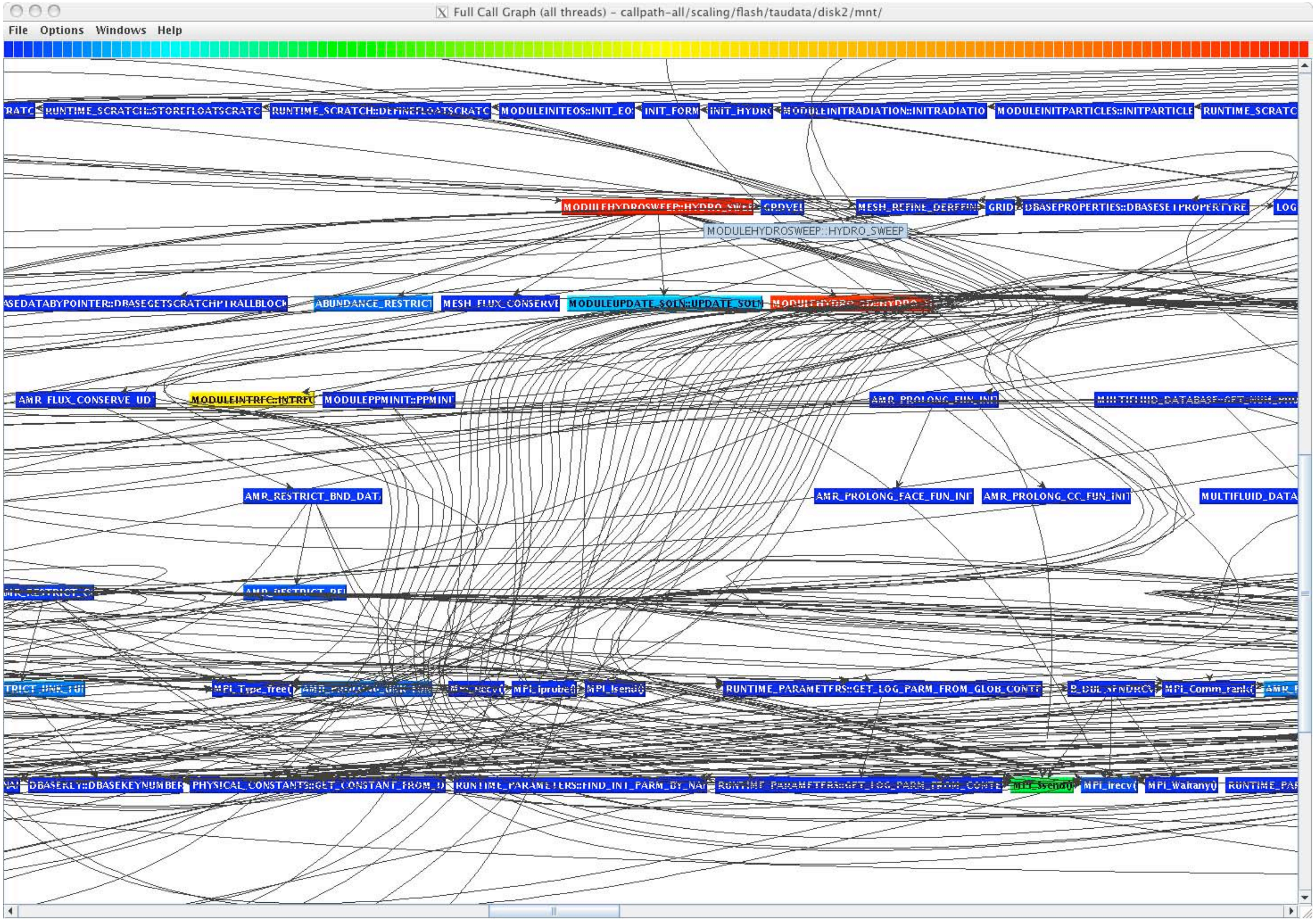
Paraprof – Highlight Callpaths



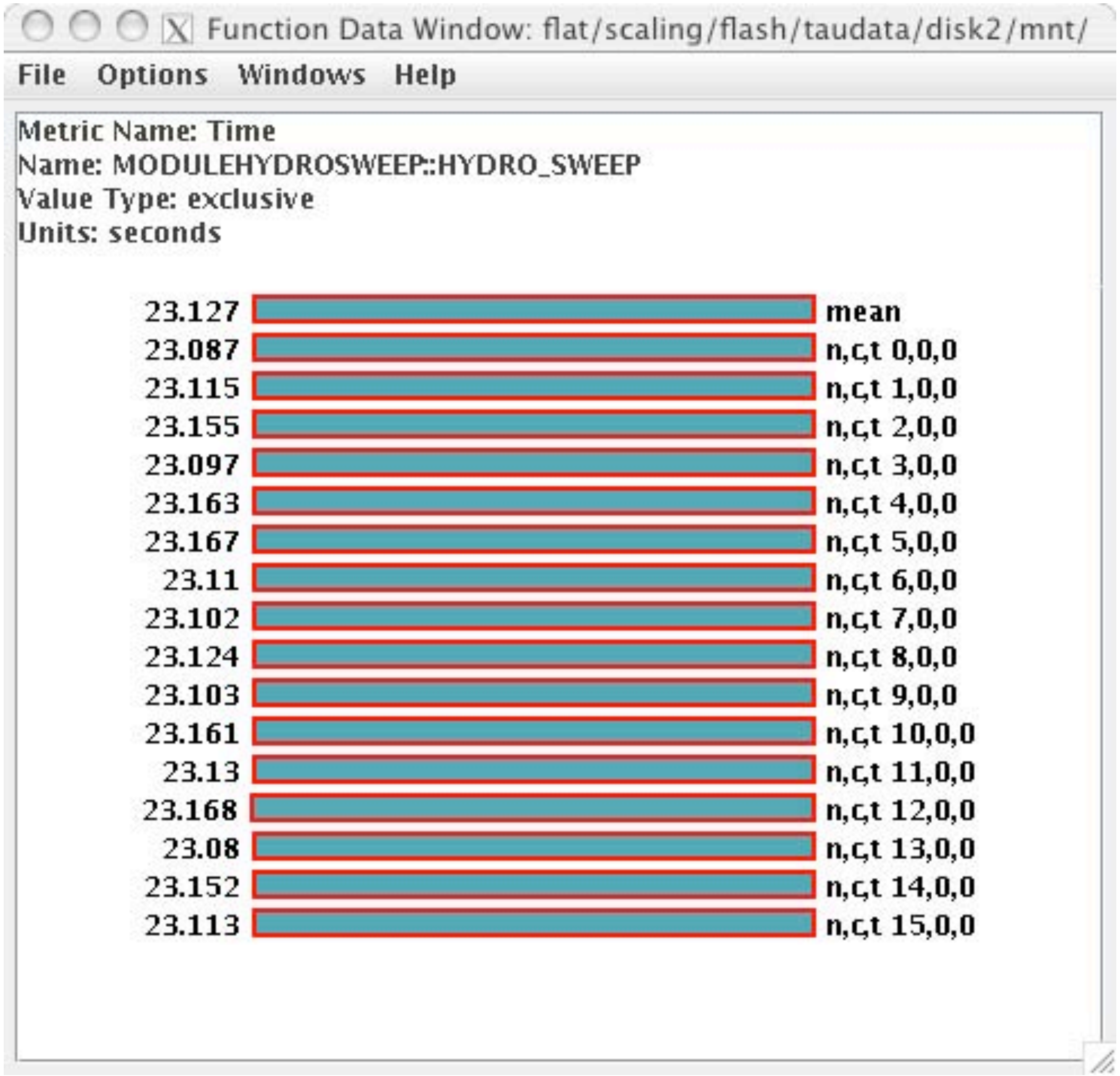
Paraprof – Callgraph View (Zoom In +/Out -)



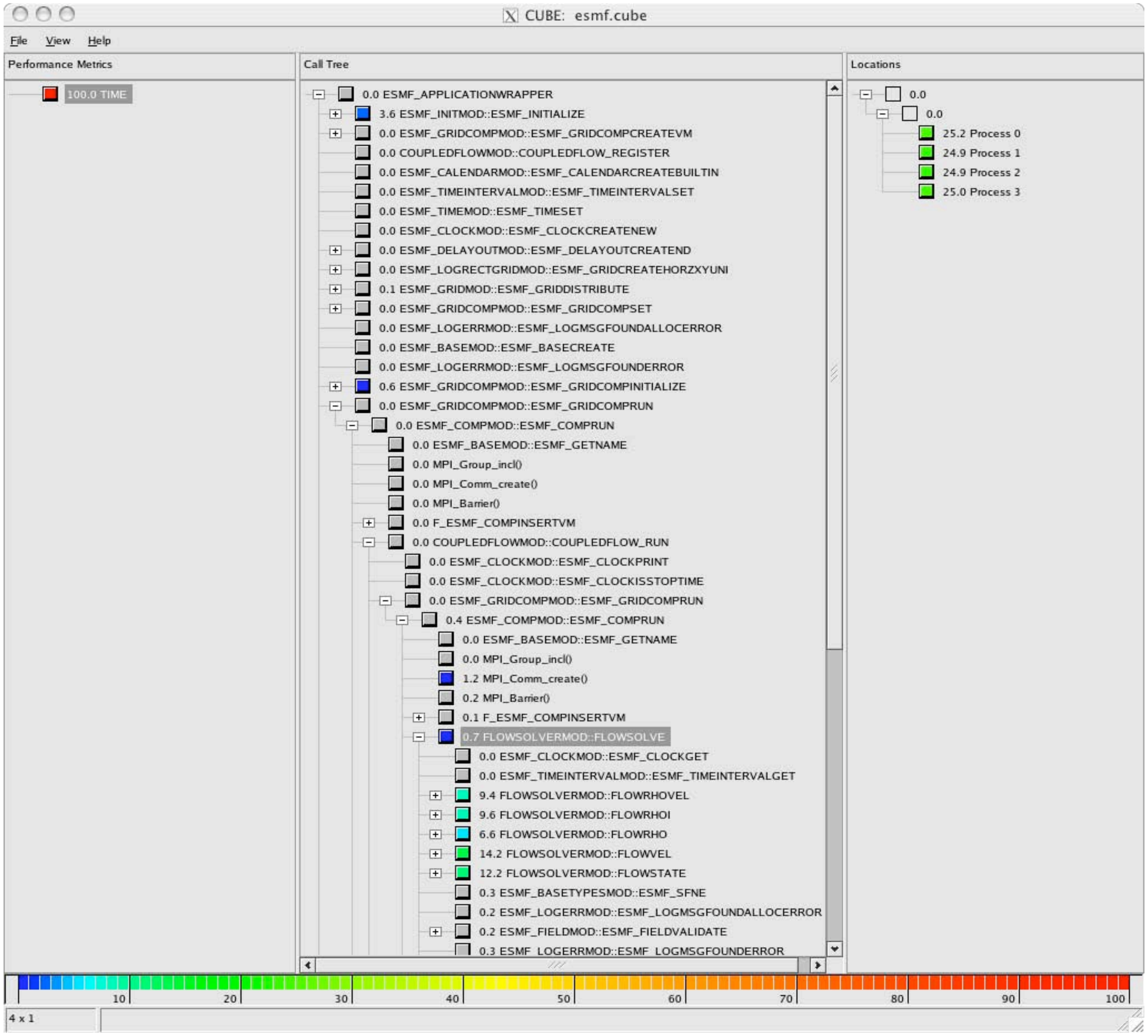
Paraprof – Callgraph View (Zoom In +/Out -)



Paraprof - Function Data Window



KOJAK's CUBE [UTK, FZJ] Browser



Linux Kernel Profiling using TAU

- Identifying points in kernel source for instrumentation
- Developing TAU's kernel profiling API
- Kernel compiled with TAU instrumentation
- Maintains per process performance data for each kernel routine
- Performance data accessible via /proc filesystem
- Instrumented application maintains data in userspace
- Performance data from application and kernel merged at program termination

Kernel Profiling Issues for IBM BlueGene/L

- I/O node kernel - Linux kernel approach
- Compute node kernel:
  - No daemon processes
  - Single address space
    - Single performance database & callstack across user/kernel
  - Keeps track of one process only (optimization)
  - Instrumented compute node kernel

TAU Performance System Status (v 2.14.2.1)

- Computing platforms (selected)
  - IBM BGL, AIX, pSeries Linux, SGI Origin, Cray RedStorm, T3E / SV-1 / X1, HP (Compaq) SC (Tru64), Sun, Hitachi SR8000, NEC SX-5/6, Linux clusters (IA-32/64, Alpha, PPC, PA-RISC, Power, Opteron), Apple (G4/5, OS X), Windows,…
- Programming languages
  - C, C++, Fortran 77/90/95, HPF, Java, OpenMP, Python
- Thread libraries
  - pthreads, SGI sproc, Java,Windows, OpenMP
- Compilers (selected)
  - IBM, Intel, Intel KAI, PGI, GNU, Fujitsu, Sun, NAG, Microsoft, SGI, Cray, HP, NEC, Absoft, Lahey

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science contracts
  - University of Utah DOE ASCI Level 1 sub-contract
  - DOE ASC/NNSA Level 3 contract
- NSF Software and Tools for High-End Computing Grant
- Research Centre Juelich
  - John von Neumann Institute for Computing
  - Dr. Bernd Mohr
- Los Alamos National Laboratory