TAU Performance System

Alan Morris, Sameer Shende, Allen D. Malony

University of Oregon

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

UNIVERSITY
OF OREGON

Acknowledgements

- Pete Beckman, ANL
- Holger Brunst and Wolfgang Nagel [TU Dresden]
- Bernd Mohr [Research Center Juelich, Germany]
- Aroon Nataraj, U. Oregon
- Suravee Suthikulpanit, U. Oregon

Outline

- Overview of features
  - Instrumentation
  - Measurement (Profiling, Tracing)
  - Analysis tools
- New features in TAU
  - Runtime MPI shared library instrumentation
  - Workload characterization
- New features for BG/L
  - PAPI now supported
  - Open Trace Format (OTF), tau2otf
  - I/O node Linux kernel profiling with TAU (KTAU)

TAU Performance System

- Tuning and Analysis Utilities (13+ year project effort)
- Performance system framework for HPC systems
  - Integrated, scalable, portable, flexible, and parallel
- Integrated toolkit for performance problem solving
  - Automatic instrumentation
  - Highly configurable measurement system with support for many flavors of profiling and tracing
  - Portable analysis and visualization tools
  - Performance data management and data mining
- http://www.cs.uoregon.edu/research/tau

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)
- 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*)
  - Runtime Linking (LD_PRELOAD)

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 $<
```

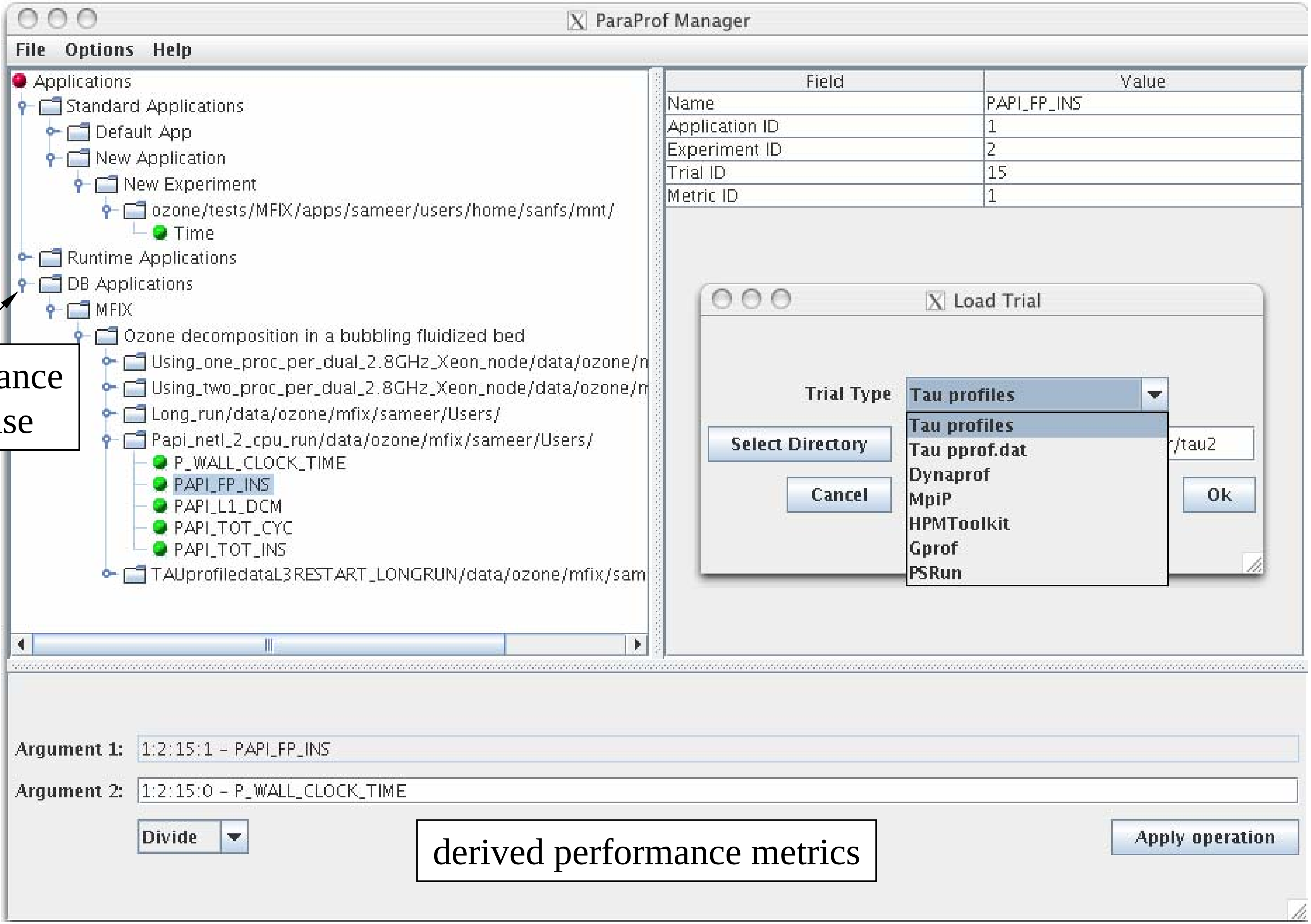
After

```
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 $<
```

Profiling Options

- Flat profiles
  - Time (or counts) spent in each routine (nodes in callgraph).
  - Exclusive/inclusive time, no. of calls, child calls
  - Support for hardware counters (PAPI, PCL), multiple counters.
- Callpath Profiles
  - Flat profiles, **plus**
  - 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.
  - Configurable callpath depth limit (`TAU_CALLPATH_DEPTH` environment variable)
- Phase based profiles
  - 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` during “`IO`” phase

ParaProf – Manager Window



Applications
- Standard Applications
  - Default App
  - New Application
    - New Experiment
      - ozone/tests/MFIX/apps/sameer/users/home/sanfs/mnt/
        - Time
- Runtime Applications
- DB Applications
  - MFIX
    - Ozone decomposition in a bubbling fluidized bed
      - Using_one_proc_per_dual_2.8GHz_Xeon_node/data/ozone/n
      - Using_two_proc_per_dual_2.8GHz_Xeon_node/data/ozone/m
      - Long_run/data/ozone/mfix/sameer/Users/
      - Papi_netl_2_cpu_run/data/ozone/mfix/sameer/Users/
        - P_WALL_CLOCK_TIME
        - PAPI_FP_INS
        - PAPI_L1_DCM
        - PAPI_TOT_CYC
        - PAPI_TOT_INS
      - TAUprofiledataL3RESTART_LONGRUN/data/ozone/mfix/sam

Field	Value
Name	PAPI_FP_INS
Application ID	1
Experiment ID	2
Trial ID	15
Metric ID	1

Trial Type: Tau profiles

- Tau profiles
- Tau pprof.dat
- Dynaprof
- MpiP
- HPMToolkit
- Gprof
- PSRun

/tau2

Argument 1: 1:2:15:1 – PAPI_FP_INS

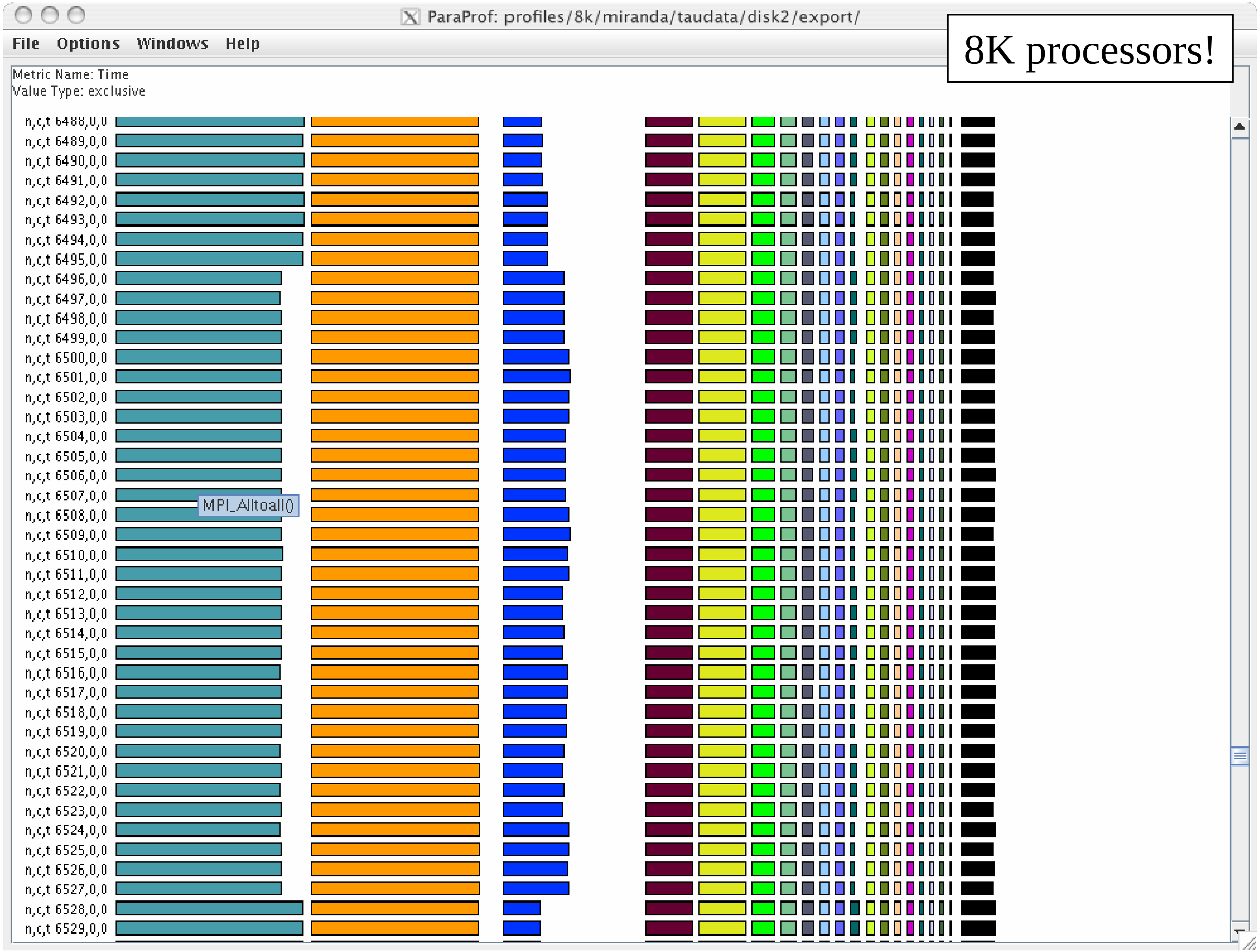
Argument 2: 1:2:15:0 – P_WALL_CLOCK_TIME

Divide

performance database

derived performance metrics

ParaProf – Full Profile (Miranda)

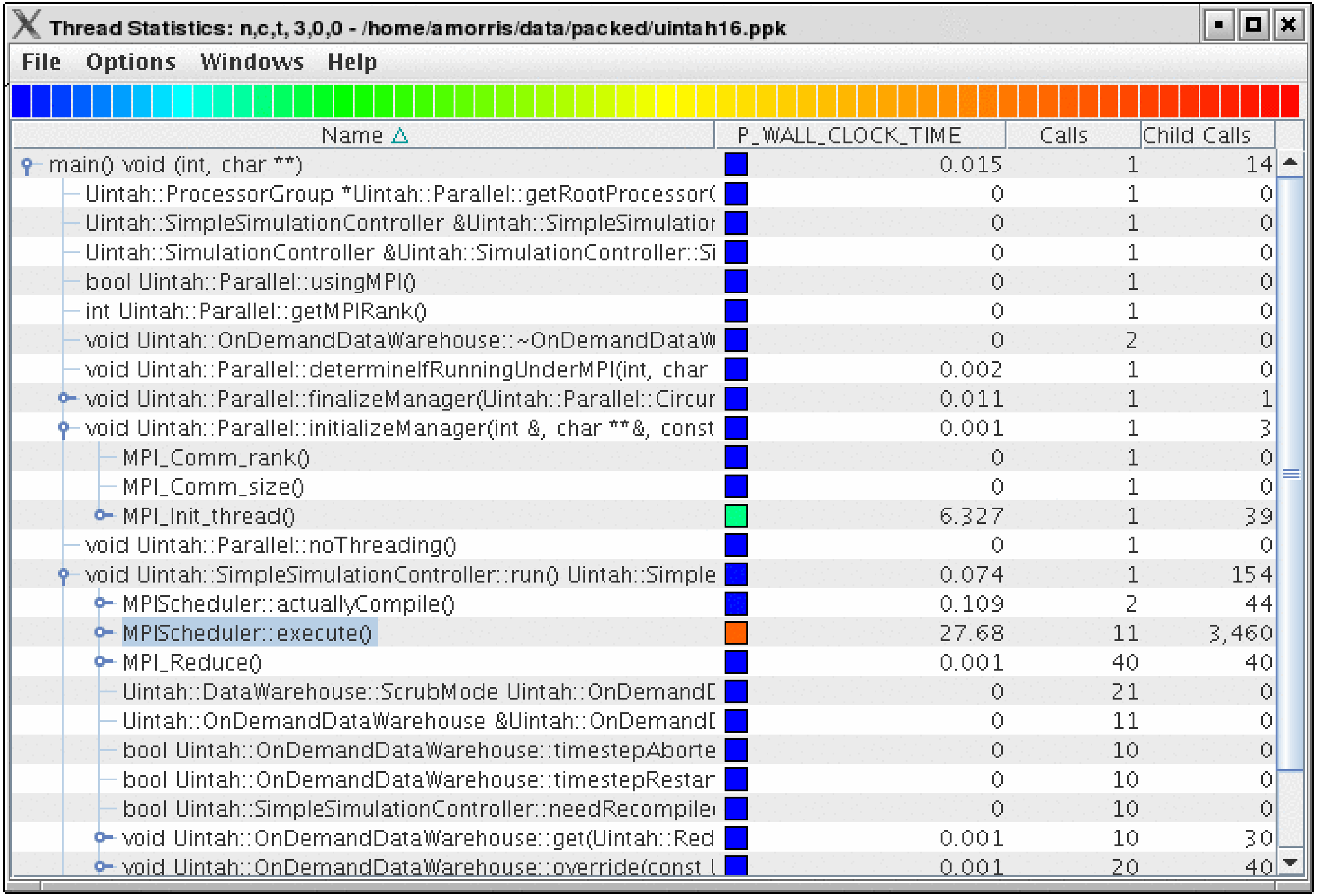


Metric Name: Time
Value Type: exclusive

```
n,c,t 6488,0,0
n,c,t 6489,0,0
n,c,t 6490,0,0
n,c,t 6491,0,0
n,c,t 6492,0,0
n,c,t 6493,0,0
n,c,t 6494,0,0
n,c,t 6495,0,0
n,c,t 6496,0,0
n,c,t 6497,0,0
n,c,t 6498,0,0
n,c,t 6499,0,0
n,c,t 6500,0,0
n,c,t 6501,0,0
n,c,t 6502,0,0
n,c,t 6503,0,0
n,c,t 6504,0,0
n,c,t 6505,0,0
n,c,t 6506,0,0
n,c,t 6507,0,0
n,c,t 6508,0,0
n,c,t 6509,0,0
n,c,t 6510,0,0
n,c,t 6511,0,0
n,c,t 6512,0,0
n,c,t 6513,0,0
n,c,t 6514,0,0
n,c,t 6515,0,0
n,c,t 6516,0,0
n,c,t 6517,0,0
n,c,t 6518,0,0
n,c,t 6519,0,0
n,c,t 6520,0,0
n,c,t 6521,0,0
n,c,t 6522,0,0
n,c,t 6523,0,0
n,c,t 6524,0,0
n,c,t 6525,0,0
n,c,t 6526,0,0
n,c,t 6527,0,0
n,c,t 6528,0,0
n,c,t 6529,0,0
```

8K processors!

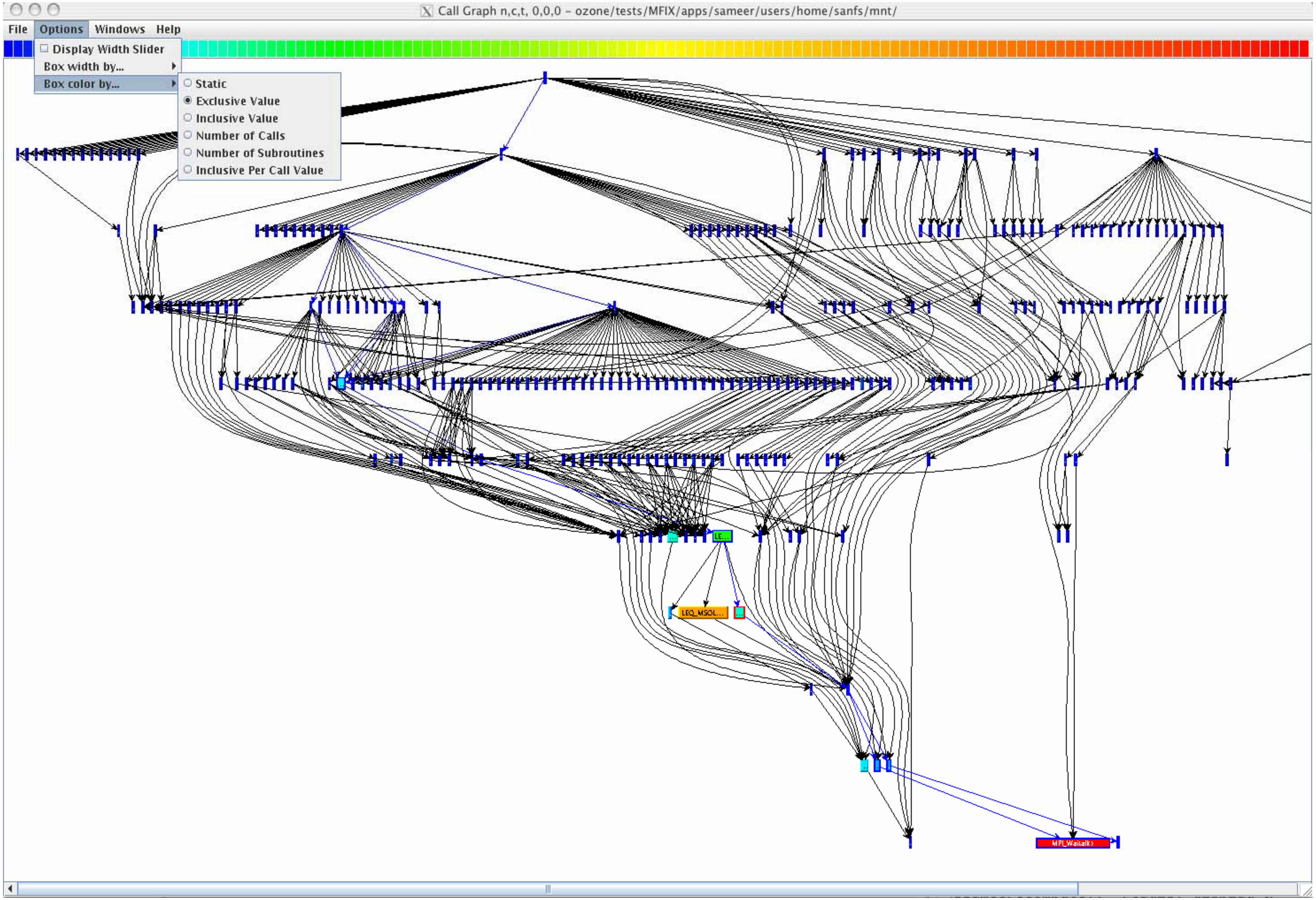
ParaProf - Statistics Table (Uintah)



Thread Statistics: n,c,t, 3,0,0 - /home/amorris/data/packed/uintah16.ppk

Name △	P_WALL_CLOCK_TIME	Calls	Child Calls
main() void (int, char **)	🟦 0.015	1	14
Uintah::ProcessorGroup *Uintah::Parallel::getRootProcessor(	🟦 0	1	0
Uintah::SimpleSimulationController &Uintah::SimpleSimulatior	🟦 0	1	0
Uintah::SimulationController &Uintah::SimulationController::Si	🟦 0	1	0
bool Uintah::Parallel::usingMPI()	🟦 0	1	0
int Uintah::Parallel::getMPIRank()	🟦 0	1	0
void Uintah::OnDemandDataWarehouse::~OnDemandDataW	🟦 0	2	0
void Uintah::Parallel::determineIfRunningUnderMPI(int, char	🟦 0.002	1	0
void Uintah::Parallel::finalizeManager(Uintah::Parallel::Circur	🟦 0.011	1	1
void Uintah::Parallel::initializeManager(int &, char **&, const	🟦 0.001	1	3
MPI_Comm_rank()	🟦 0	1	0
MPI_Comm_size()	🟦 0	1	0
MPI_Init_thread()	🟩 6.327	1	39
void Uintah::Parallel::noThreading()	🟦 0	1	0
void Uintah::SimpleSimulationController::run() Uintah::Simple	🟦 0.074	1	154
MPIScheduler::actuallyCompile()	🟦 0.109	2	44
MPIScheduler::execute()	🟧 27.68	11	3,460
MPI_Reduce()	🟦 0.001	40	40
Uintah::DataWarehouse::ScrubMode Uintah::OnDemandD	🟦 0	21	0
Uintah::OnDemandDataWarehouse &Uintah::OnDemandD	🟦 0	11	0
bool Uintah::OnDemandDataWarehouse::timestepAborte	🟦 0	10	0
bool Uintah::OnDemandDataWarehouse::timestepRestar	🟦 0	10	0
bool Uintah::SimpleSimulationController::needRecompile	🟦 0	10	0
void Uintah::OnDemandDataWarehouse::get(Uintah::Red	🟦 0.001	10	30
void Uintah::OnDemandDataWarehouse::override(const U	🟦 0.001	20	40

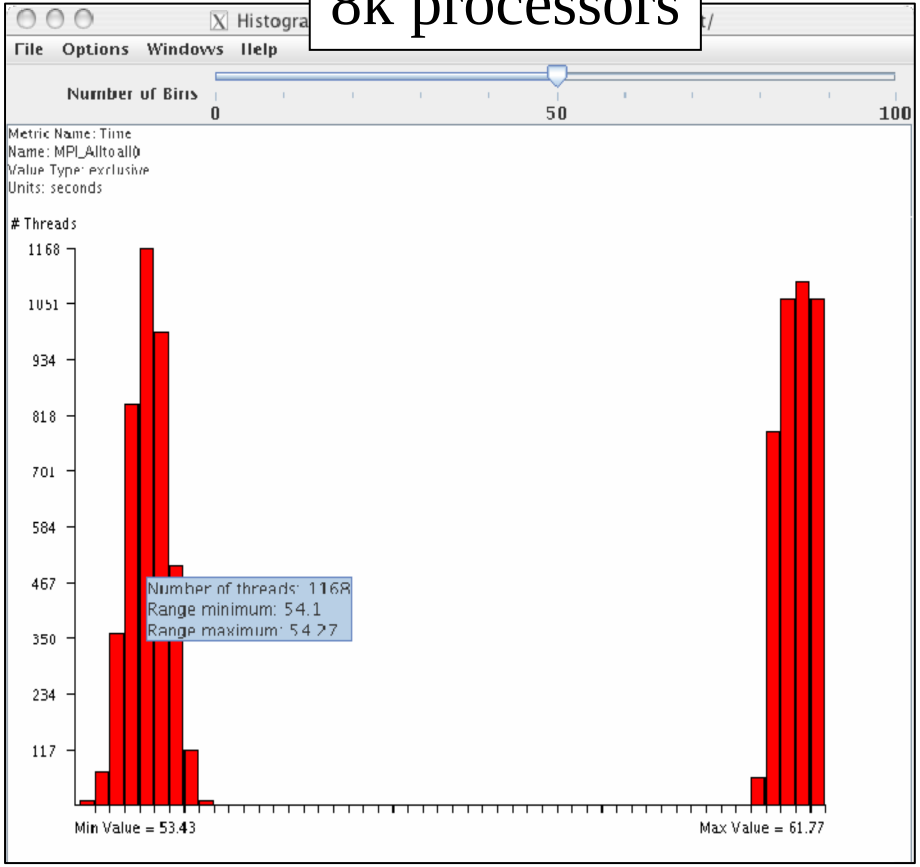
ParaProf –Callgraph View (MFIX)



ParaProf – Histogram View (Miranda)

- Scalable 2D displays

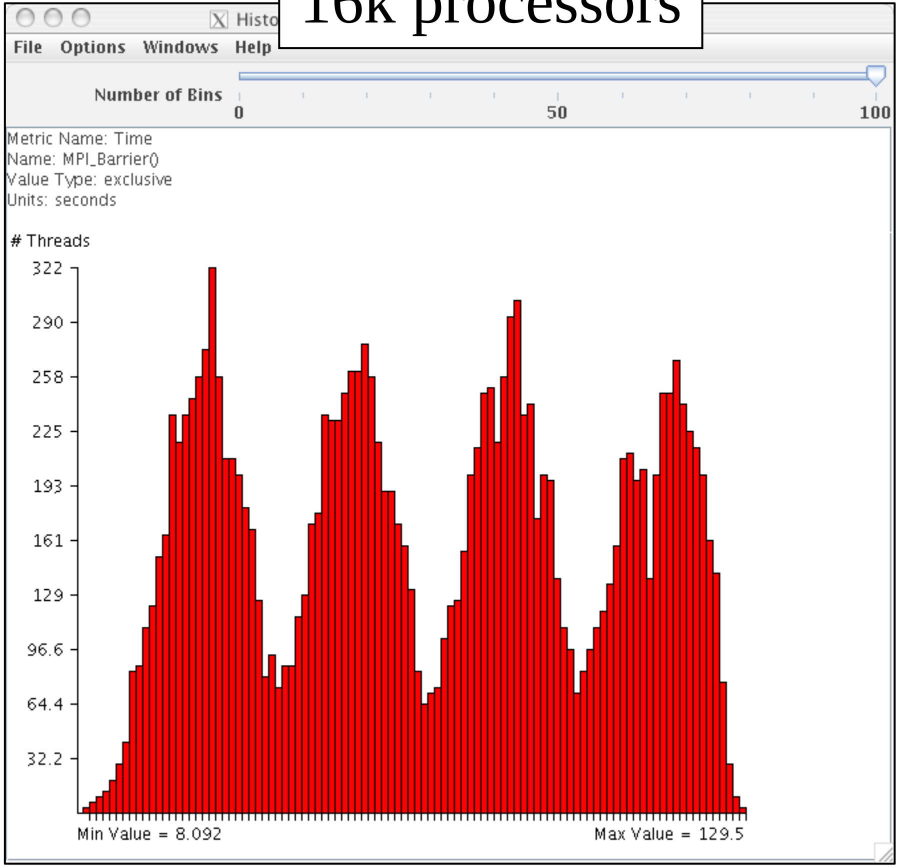
8k processors



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

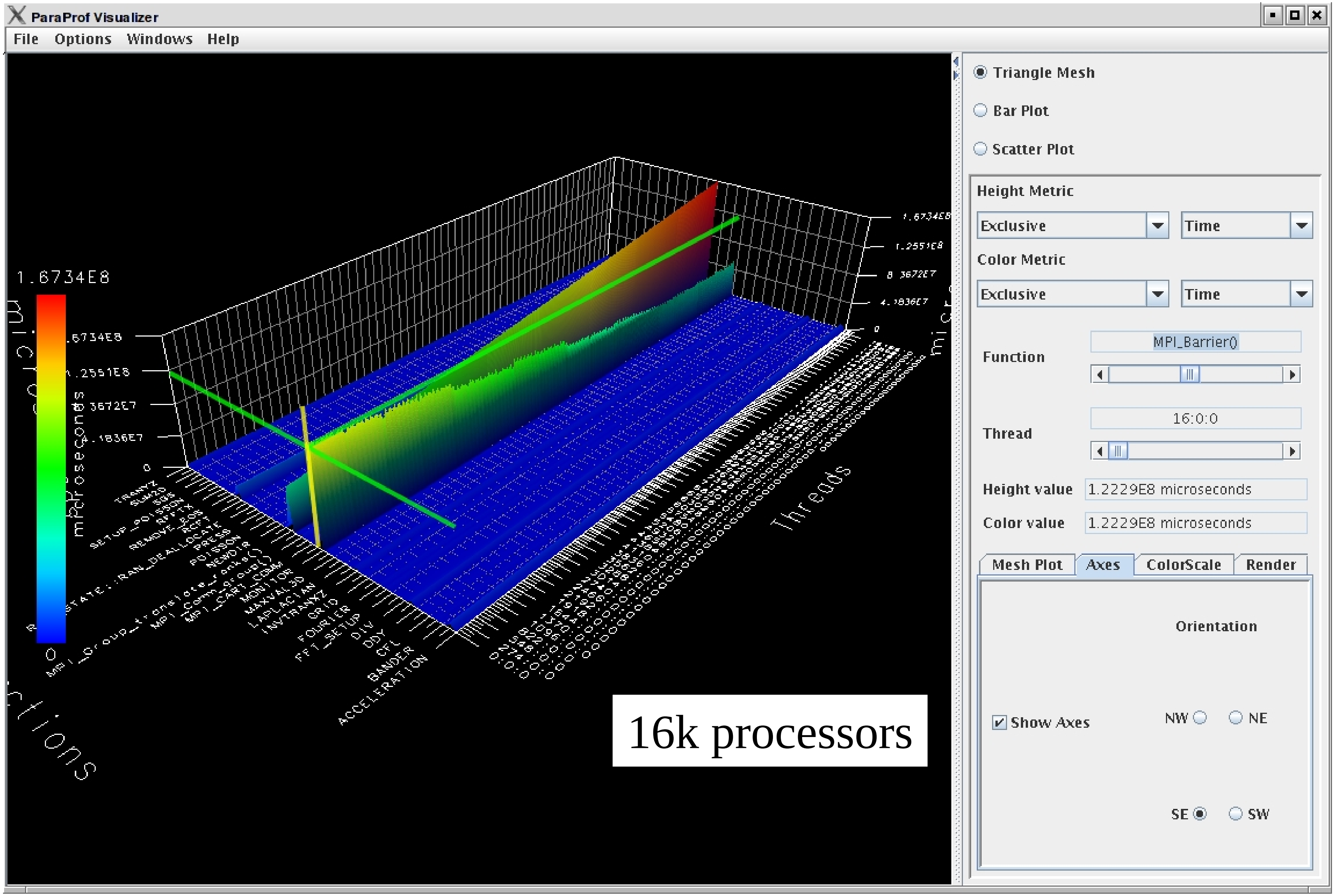
Number of threads: 1168
Range minimum: 54.1
Range maximum: 54.27

16k processors



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

ParaProf – 3D Full Profile (Miranda)



16k processors

● 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

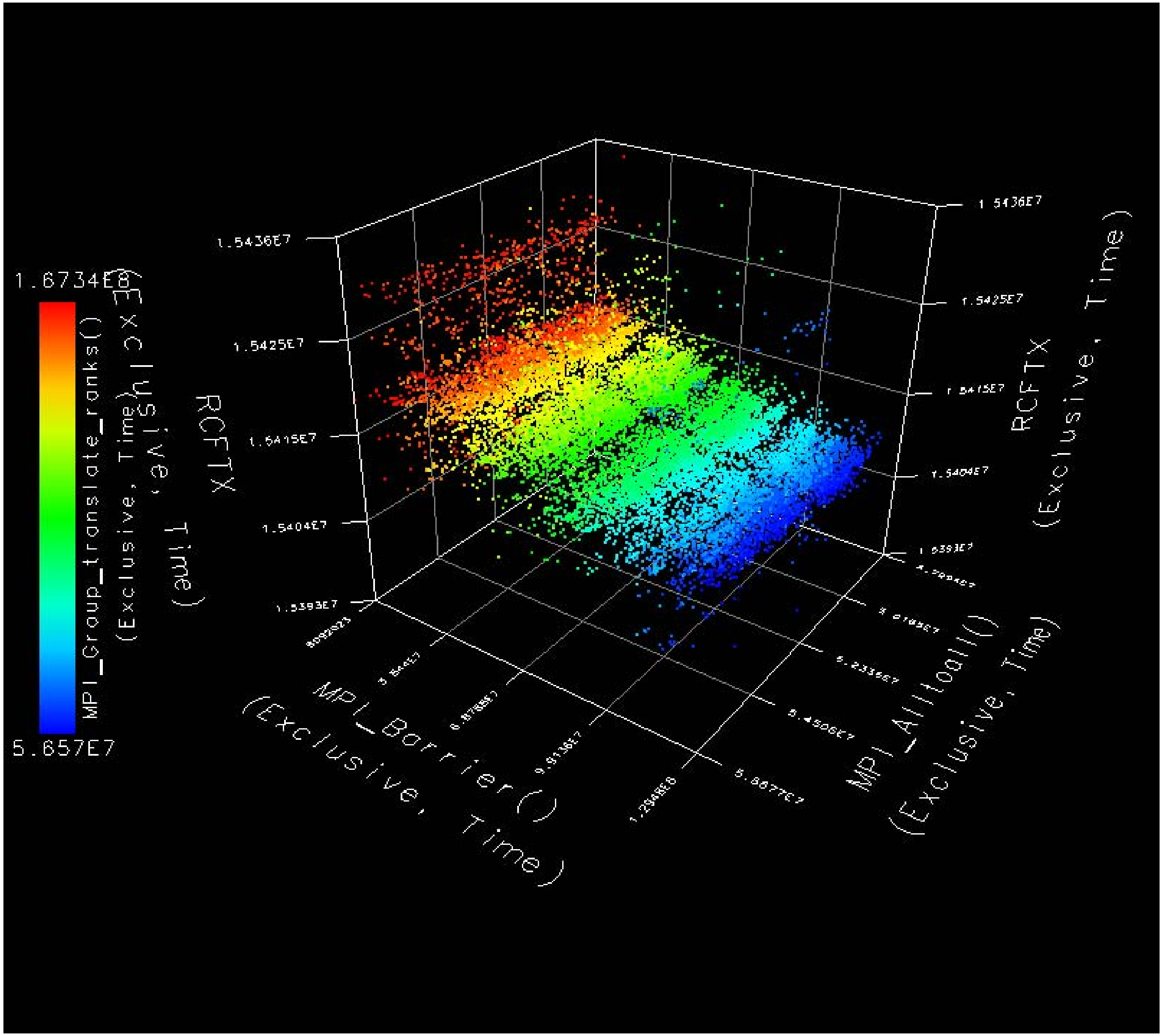
☑ Show Axes

NW ○ ○ NE

SE ● ○ SW

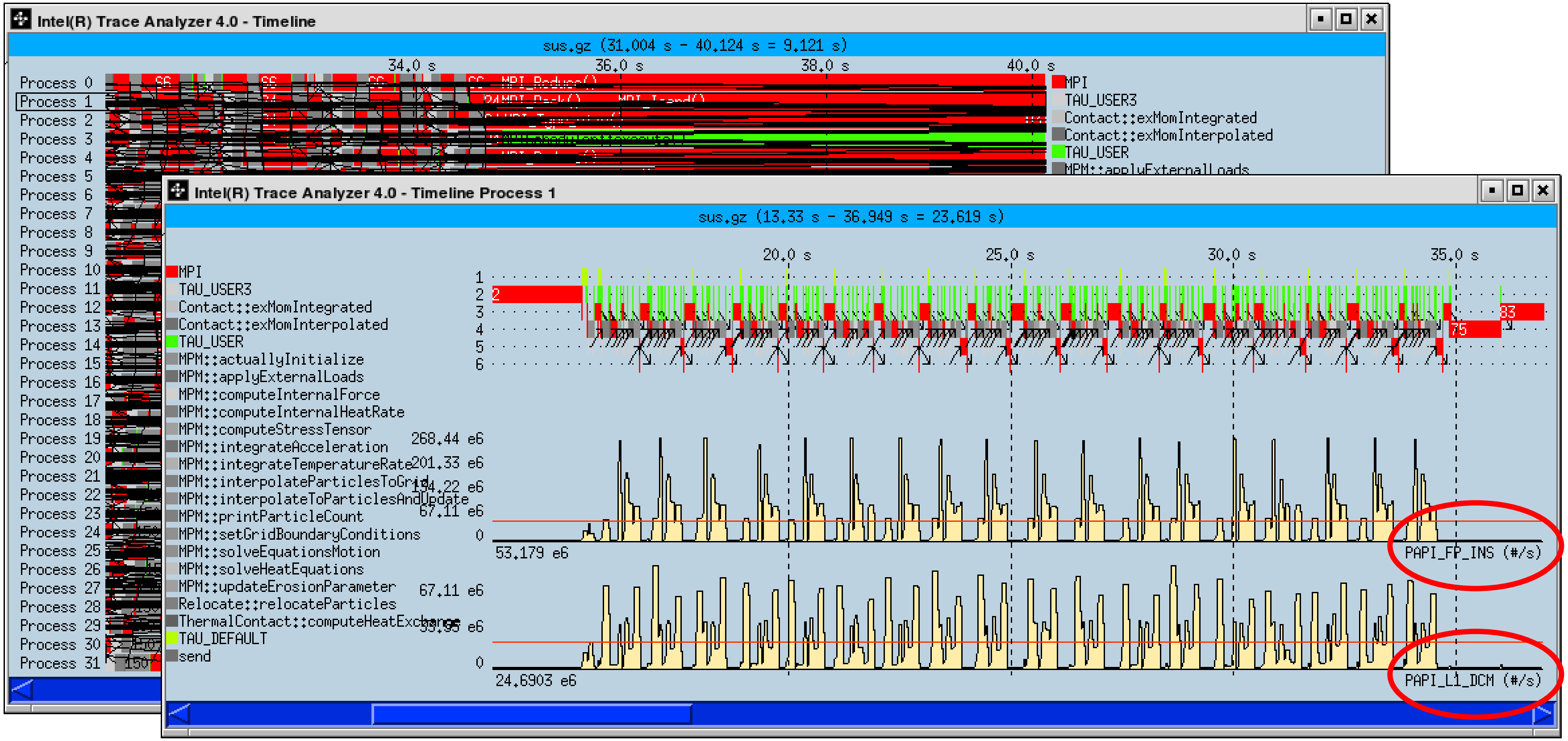
ParaProf – 3D Scatterplot (Miranda)

- Each point is a "thread" of execution
- Relation between four routines shown at once



Tracing (Vampir)

- Trace analysis provides in-depth understanding of temporal event and message passing relationships
- Traces can even store hardware counters



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
- Soon on AIX using `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

Workload Characterization

- 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(int 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

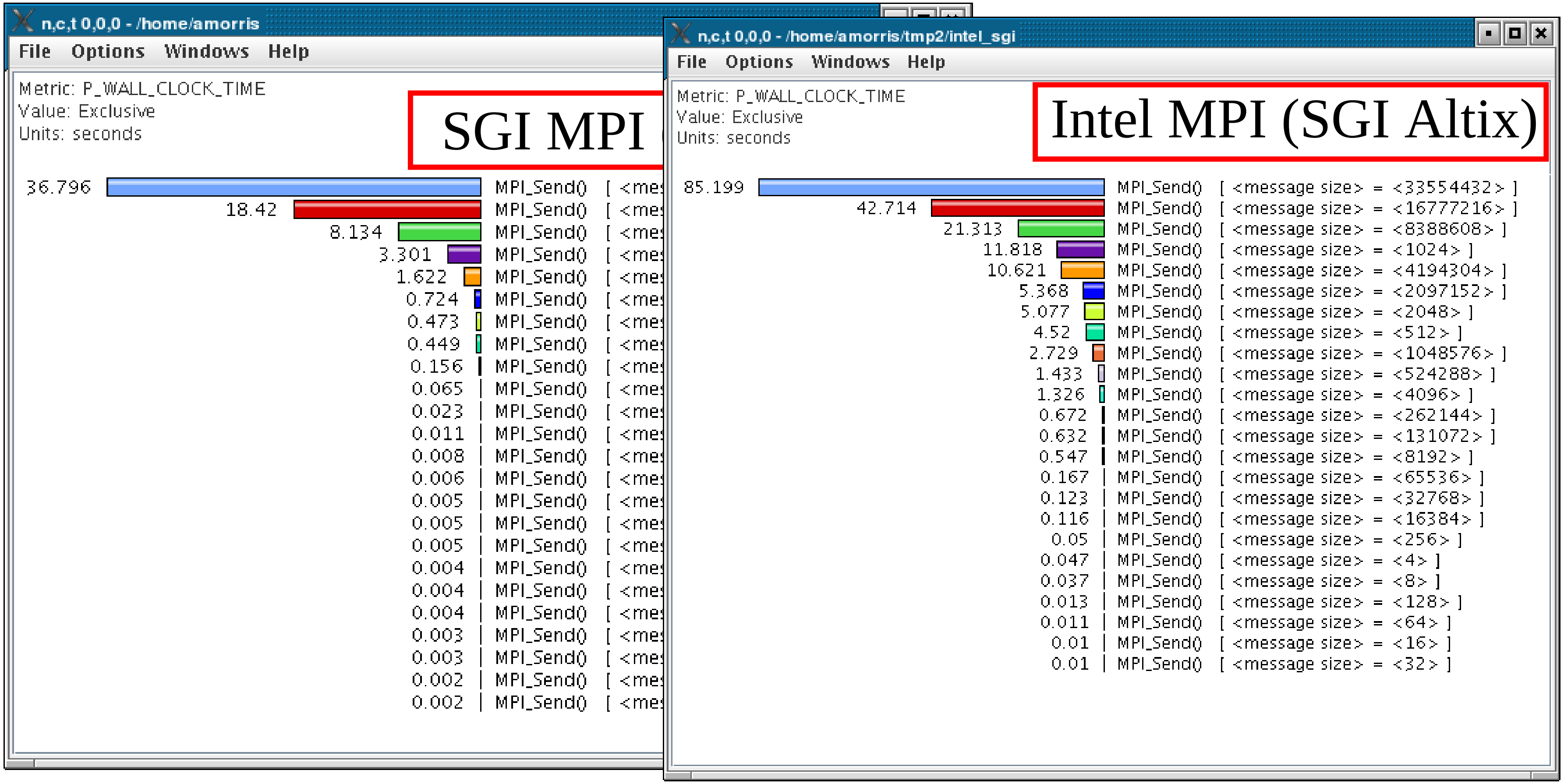
- Simple example, send/receive squared message sizes (0-32MB)

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv) {
  int rank, size, i, j;
  int buffer[16*1024*1024];
  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<16*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

- ❒ Use `tau_load.sh` to instrument MPI routines (SGI Altix)

```
% icc mpi.c –lmpi

% mpirun –np 2 tau_load.sh a.out
```



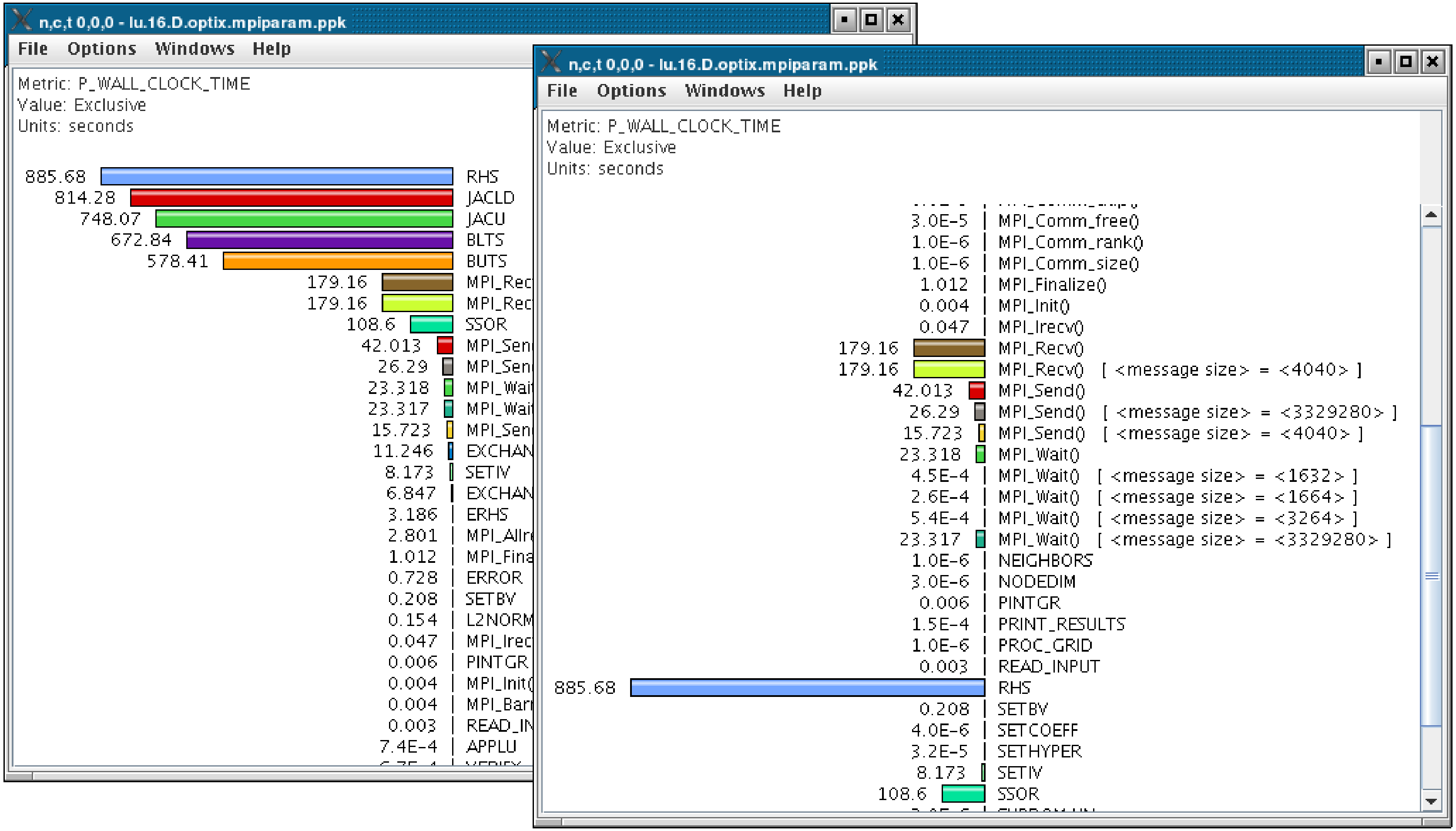
n,c,t 0,0,0 - /home/amorris
File Options Windows Help
SGI MPI
n,c,t 0,0,0 - /home/amorris/tmp2/intel_sgi
File Options Windows Help
Intel MPI (SGI Altix)

```
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

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

```
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 16 processors of SGI Altix)



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

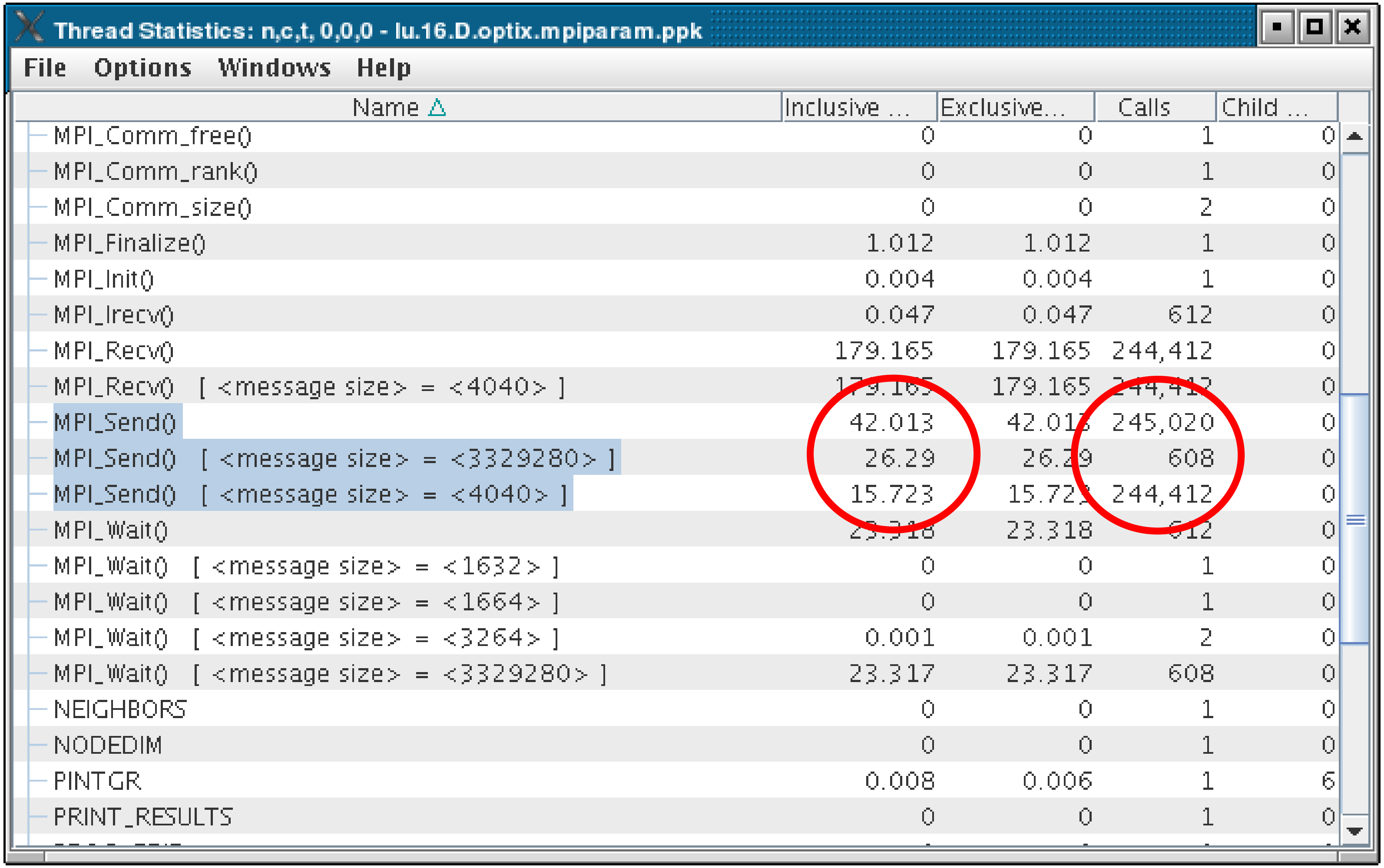
```
Metric: P_WALL_CLOCK_TIME
Value: Exclusive
Units: seconds

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
```

Workload Characterization

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



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

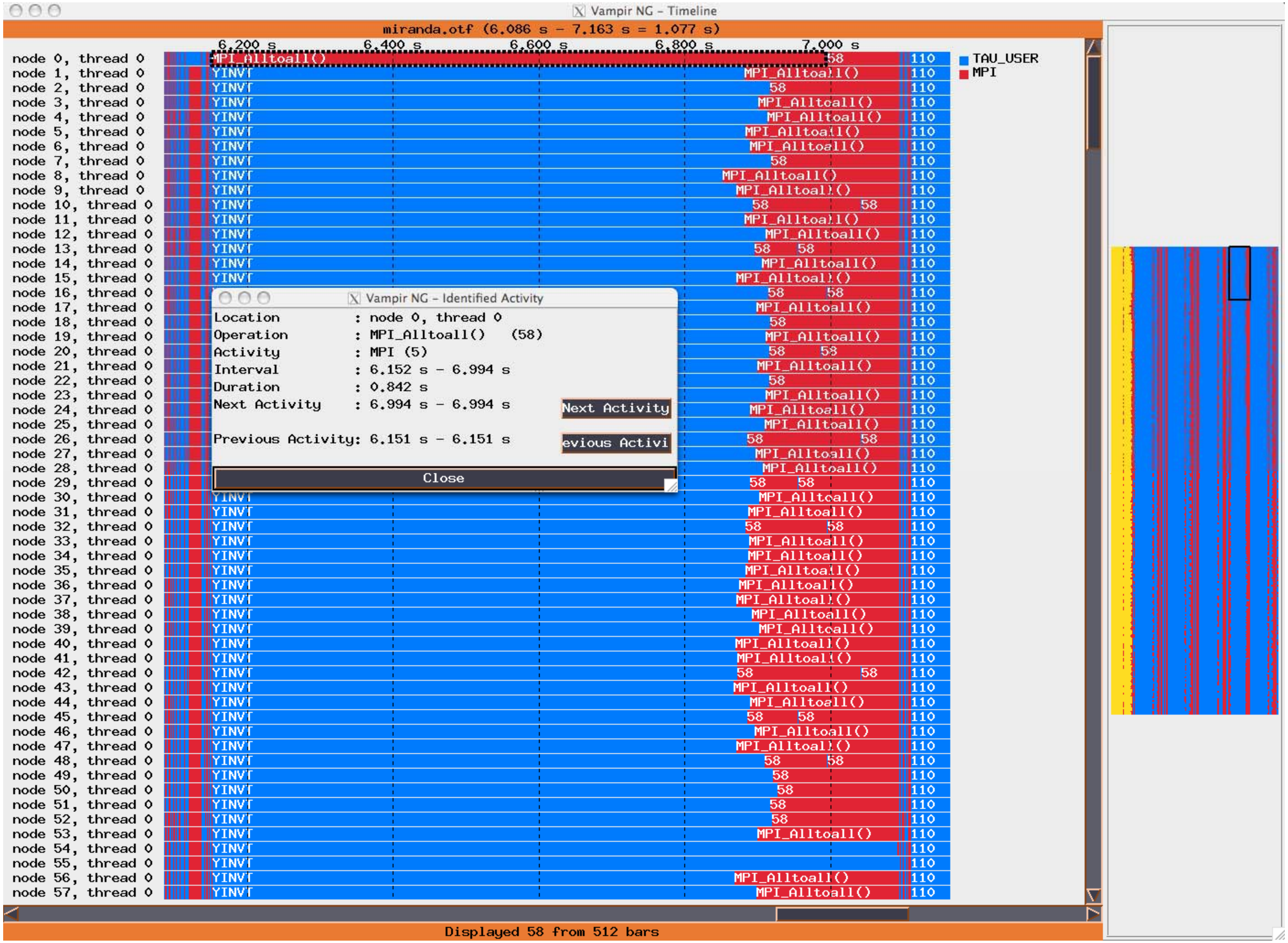
Name △	Inclusive ...	Exclusive...	Calls	Child ...
MPI_Comm_free()	0	0	1	0
MPI_Comm_rank()	0	0	1	0
MPI_Comm_size()	0	0	2	0
MPI_Finalize()	1.012	1.012	1	0
MPI_Init()	0.004	0.004	1	0
MPI_Irecv()	0.047	0.047	612	0
MPI_Recv()	179.165	179.165	244,412	0
MPI_Recv() [<message size> = <4040>]	179.165	179.165	244,412	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

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)
- Open Trace Format (OTF)
  - Hierarchical trace format, efficient streams based parallel access with VNGD
  - Replacement for proprietary formats such as STF
  - Tracing library available on IBM BG/L platform
  - Open Source release of OTF by SC06
- Development of OTF supported by LLNL contract

http://www.vampir-ng.de

VNG Timeline Display (Miranda on BGL)



Vampir NG – Timeline

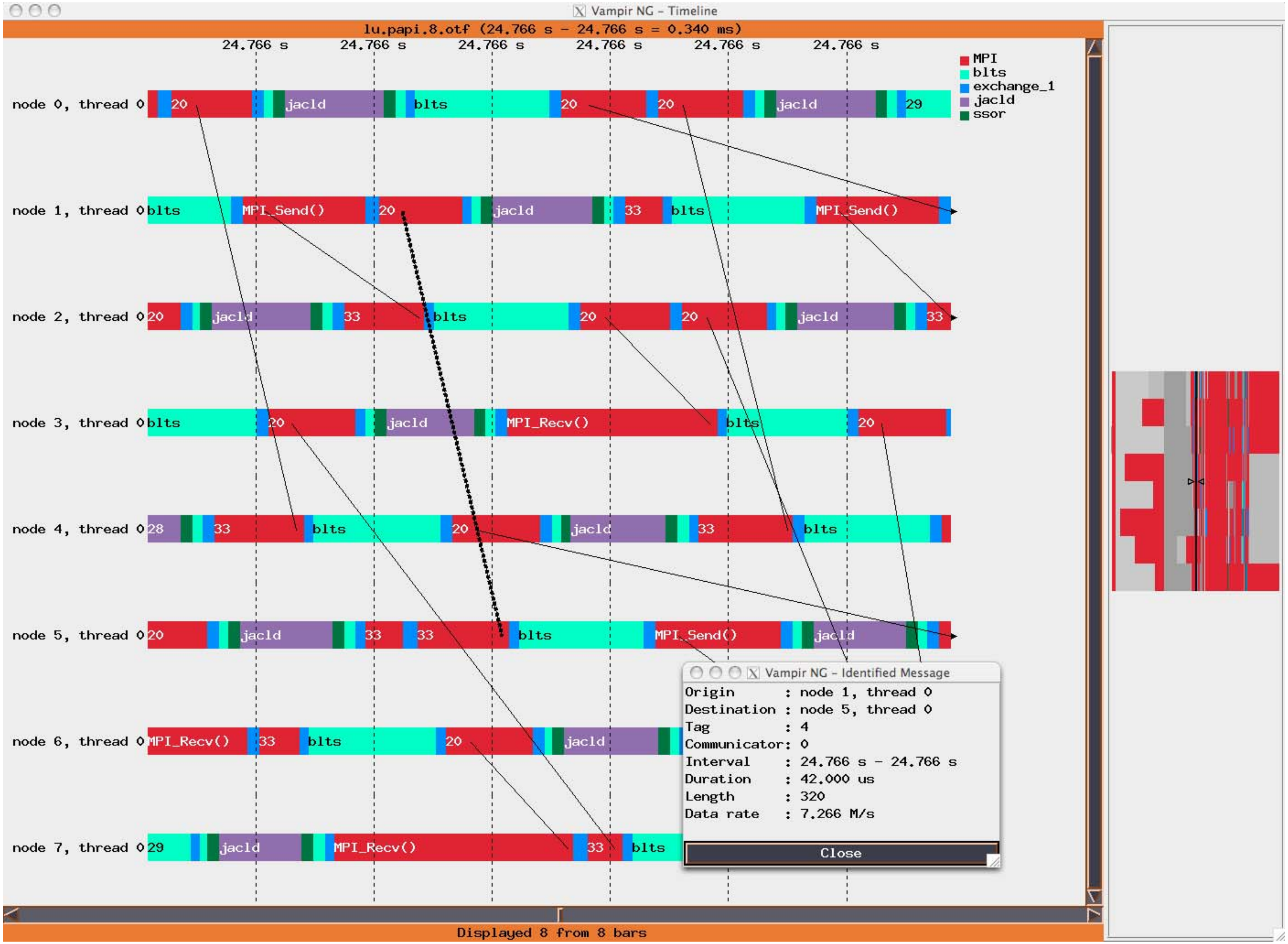
miranda.otf (6.086 s – 7.163 s = 1.077 s)

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
```

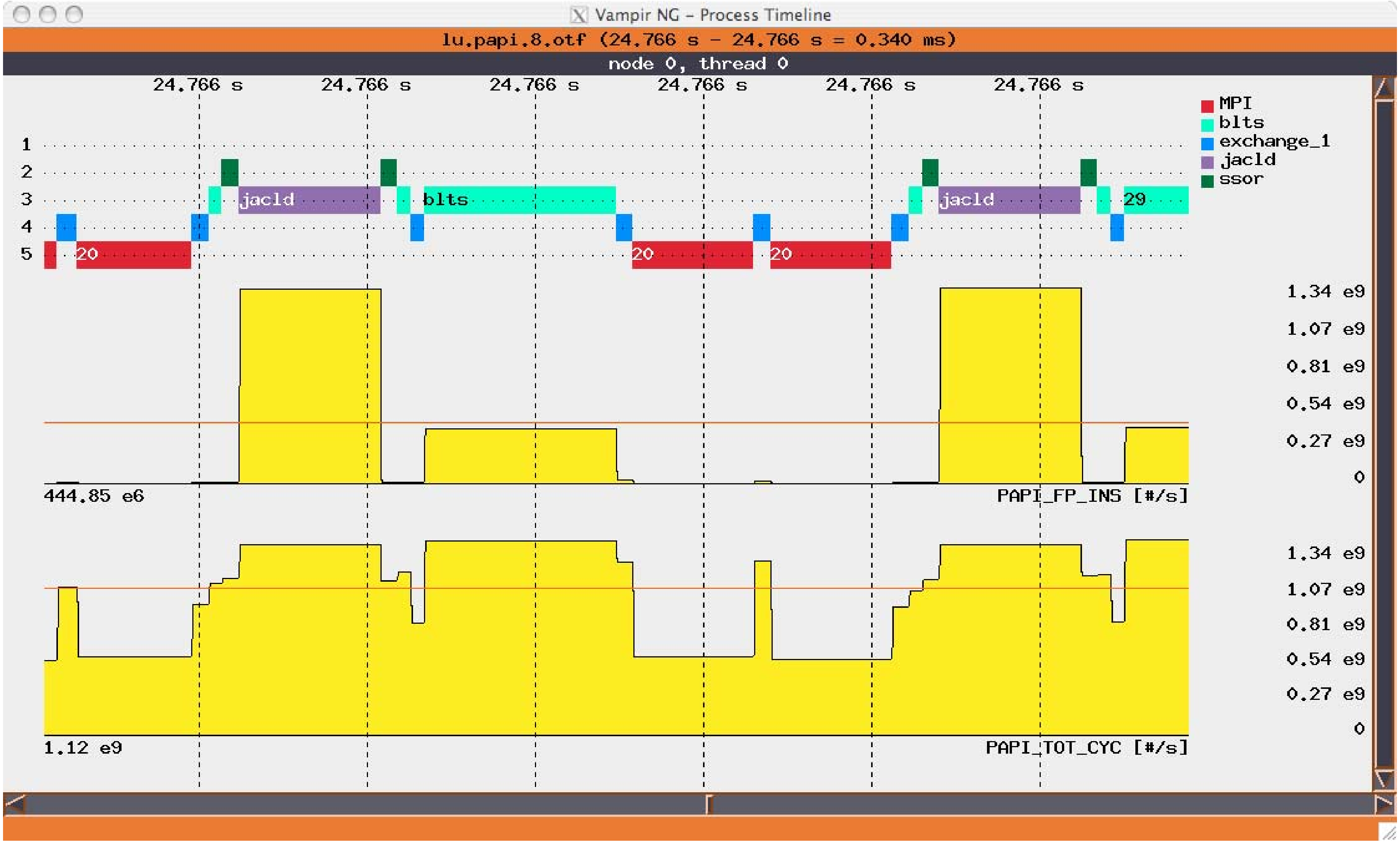
Displayed 58 from 512 bars

VNG Timeline Zoomed In



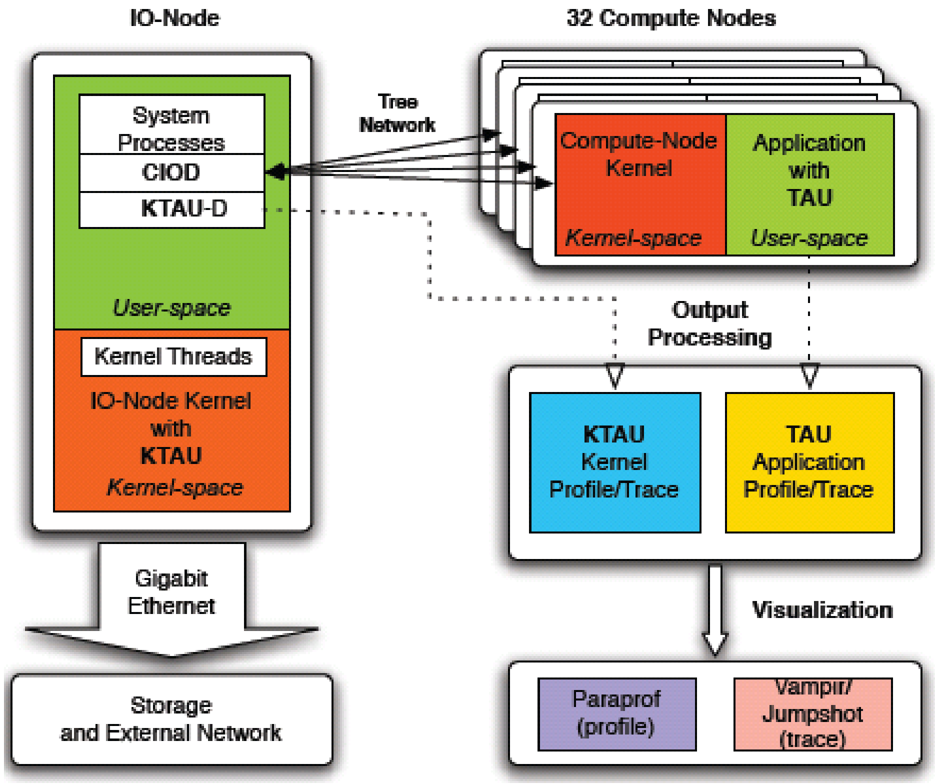
```
Origin      : node 1, thread 0
Destination : node 5, thread 0
Tag         : 4
Communicator: 0
Interval    : 24.766 s – 24.766 s
Duration    : 42.000 us
Length      : 320
Data rate   : 7.266 M/s
```

VNG Process Timeline with PAPI Counters



KTAU on BG/L

- KTAU designed for Linux Kernel profiling
- Provides merged application/system profile
- Runs on I/O-Node of BG/L

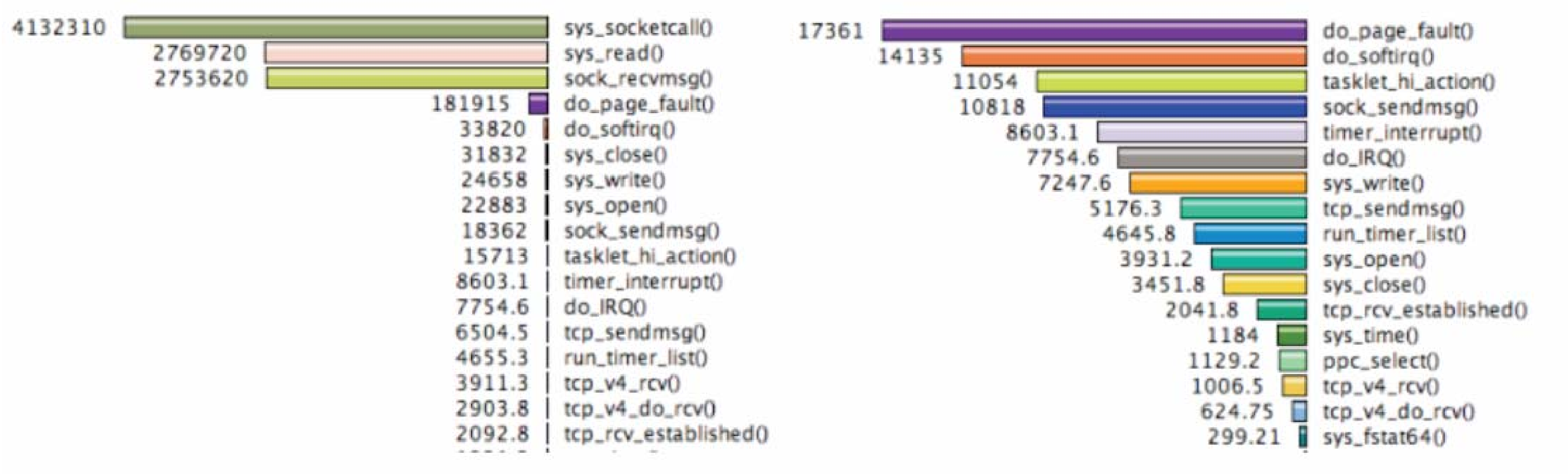


KTAU on BG/L

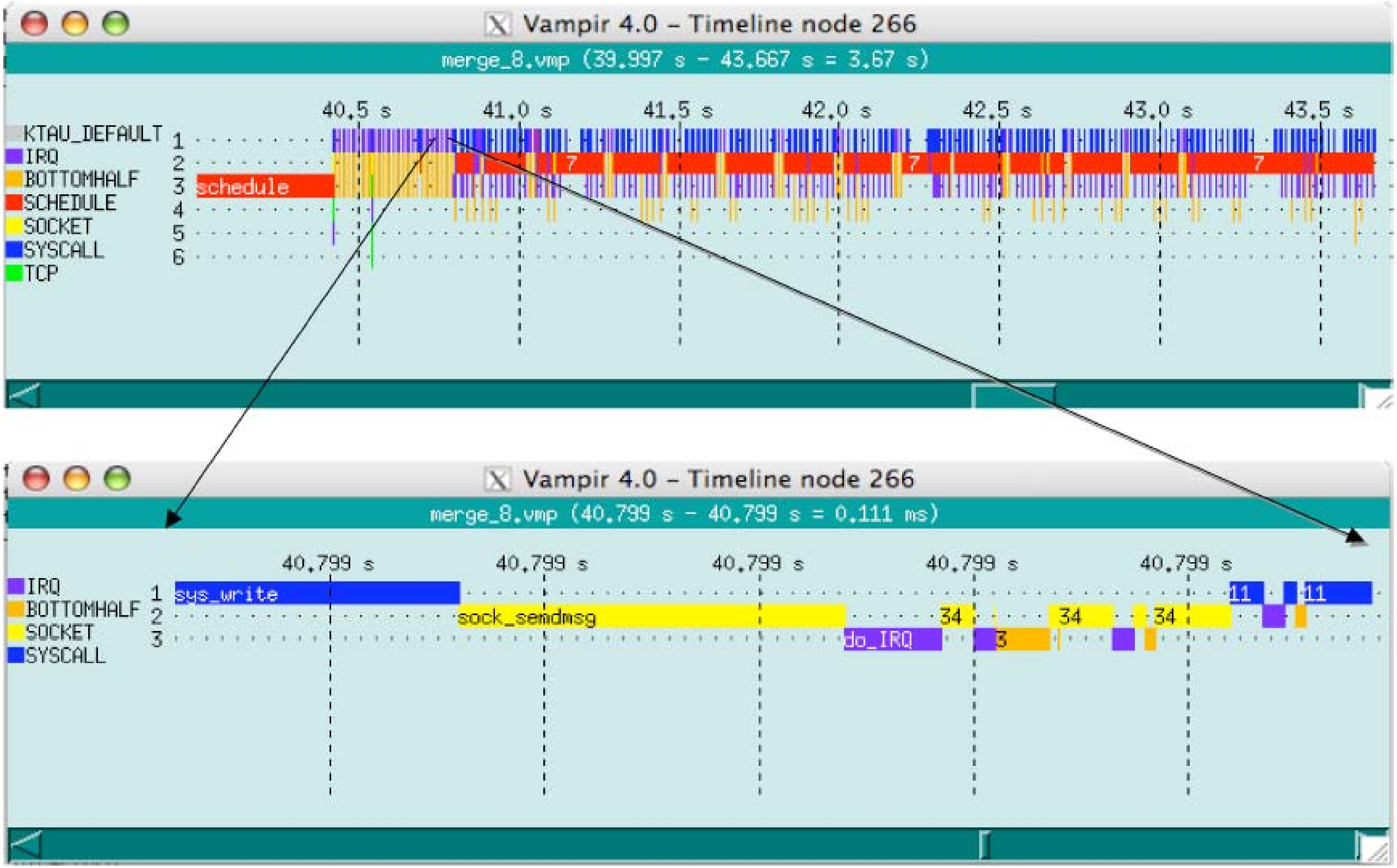
- Current status
  - Detailed I/O Node kernel profiling/tracing
  - KTAU integrated into ZeptoOS build system
  - KTAU-Daemon (KTAU-D) on I/O Node
    - Monitors system-wide and/or individual processes
  - Visualization of trace/profile of ZeptoOS and CIOD
    - Vampir/JumpShot (trace), and Paraprof (profile)

KTAU on BG/L

- Example of I/O Node profile data
- Numbers in microseconds, inclusive left, exclusive right



KTAU on BG/L, Trace Data



Vampir 4.0 – Timeline node 266

merge_8.vmp (39.997 s - 43.667 s = 3.67 s)

Vampir 4.0 – Timeline node 266

merge_8.vmp (40.799 s - 40.799 s = 0.111 ms)

Support Acknowledgements

- Department of Energy (DOE)
  - Office of Science contracts
  - University of Utah ASC Level 1 sub-contract
  - LLNL ASC/NNSA Level 3 contract
  - LLNL ParaTools/GWT contract
- NSF
  - High-End Computing Grant
- T.U. Dresden, GWT
  - Dr. Wolfgang Nagel and Holger Brunst
- Research Centre Juelich
  - Dr. Bernd Mohr
- Los Alamos National Laboratory contracts