MPI Performance Tools

Physics 244
James Bordner

31 May 2012

Outline

1. Introduction
2. Timing functions: `MPI_Wtime,etime,gettimeofday`
3. Profiling tools
   - time: `gprof,TAU`
   - hardware counters: `PAPI,PerfSuite,TAU`
   - MPI communication: `IPM,TAU`
4. MPI tracing tools
   - MPI communication: `MPE+Jumpshot`
5. Summary

Why worry about performance?

- *"Achieving 10% of the peak performance on a distributed parallel machine in a real application is usually considered an accomplishment"*[1]
- *"Applying optimization techniques...can significantly speed up a program, often by factors of 10 to 100"*[1]
- Performance tools invaluable for diagnosing performance problems!

[1] "Performance Optimization of Numerically Intensive Codes"—Stefan Goedecker, Adolfy Hoisie

Steps to improve performance

1. Measure performance
   - Select performance tool
   - Run application with appropriate data set
   - Obtain performance data files
2. Analyze performance
   - Where is code inefficient?
   - Why is code inefficient?
3. Optimize performance
   - Experiment with compiler optimization flags
   - Restructure code
   - Rearrange data
   - Improve MPI communication
   - Change/modify algorithms

Performance tool characteristics

- Different tools have different trade-offs
  - *specific* versus *general*
  - *small-scale* versus *large-scale*
  - *automatic* versus *instrumented*
  - *profiling* versus *tracing*
- Different tools have different advantages
  - ease of use
  - performance data file size
  - availability

Performance tool categories

- Timing functions
  - Manually time regions of code
- Profiling tools
  - Average performance of functions or application
  - Time, flops, cache misses, MPI communication, etc.
  - Tools: `gprof`, `TAU`, `PAPI`, `PerfSuite`, `IPM`
- Tracing MPI communication
  - Tracing tools log every call to MPI
  - Can view dynamic communication behavior
  - Typically large trace files
  - Tools: `MPE/Jumpshot`

Timers

Timers are conceptually the simplest performance tools

- The "Hello world" of performance measurement
- Widely available
- Limited usefulness
  - Require manual instrumentation
  - More work for larger programs
  - Tells you how much time spent where, but not why

Timing MPI programs

MPI_Wtime()

In MPI programs, `MPI_Wtime()` returns time in seconds

```
#include <mpi.h>

double t,time;

t = MPI_Wtime();

/* code to time */

time = MPI_Wtime() - t;
```

The timer resolution can be obtained by calling `MPI_Wtick()`

Timing Fortran code

etime()

The Fortran function etime returns elapsed time in seconds

```
real time, tarray(2), etime

time = etime(tarray)

!  code to time

time = etime(tarray) - time
```

tarray(1) contains user time in seconds
tarray(2) contains system time in seconds

Timing C/C++ code

gettimeofday()

The C function gettimeofday() returns a time in s and μs

```
#include <sys/time.h>

struct timeval t1, t2;
struct timezone tz;
double time;

gettimeofday(&t1, &tz);

/* code to time */

gettimeofday(&t2, &tz);
time = (t2.tv_sec-t1.tv_sec) + 1e-6*(t2.tv_usec-t1.tv_usec);
```

Profiling

- Profiling tools summarize performance over entire run
- Typically at the function level
- May measure time, hardware counters, or MPI communication
  - `gprof` can profile in terms of time
  - `PerfSuite(PAPI)` can profile in terms of hardware counters
  - `IPM, TAU` can profile in terms of MPI communication

The gprof profiler

gprof is a widely available profiler

- Works by regularly sampling the program counter
- Automatic: no instrumentation necessary
- Not designed for parallel codes
- Compile and link application with `-g -pg` flags
- Running application generates a data file `gmon.out`
- Post-processing with `gprof a.out [gmon.out]` generates readable report

gprof flat profile

. . .

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
27.19	1.18	1.18	1140	0.00	0.00	twoshock_
13.82	1.78	0.60	1140	0.00	0.00	inteuler_
10.83	2.25	0.47	1140	0.00	0.00	euler_
10.37	2.70	0.45	259920	0.00	0.00	intvar_
7.37	3.02	0.32				H5C_flush_cache
5.07	3.24	0.22				H5C_create
3.69	3.40	0.16	2280	0.00	0.00	pgas2d_dual_
1.84	3.48	0.08	10	0.01	0.01	dep_grid_cic_
1.15	3.53	0.05	8552	0.00	0.00	copy3d_
1.15	3.58	0.05	80	0.00	0.00	grid::CopyBaryonFieldToOldBaryonField()

. . .

gprof call graph

```
                    .                       .                       .
                    .                       .                       .
                    .                       .                       .

index % time    self  children    called     name
-----------------------------------------------
                0.00    3.57       1/1           main [1]
[2]     82.3    0.00    3.57       1         EvolveHierarchy(...) [2]
                0.00    3.39      10/10          EvolveLevel(...) [3]
                0.04    0.04     130/130         grid::ComputeTimeStep() [20]
                0.00    0.06      11/11          RebuildHierarchy(...) [27]
                0.00    0.02      11/11          CheckForOutput(...) [72]
                0.00    0.02       1/2           Group_WriteAllData(...) [49]
                0.00    0.01      16/16          CopyOverlappingZones(...) [100]
                0.00    0.00       8/168         grid::SetExternalBoundaryValues(...) [64]
                0.00    0.00       1/111         CommunicationReceiveHandler(...) [74]
                0.00    0.00      44/96          PrintMemoryUsage(char*) [2948]
                0.00    0.00      41/2438        ReturnWallTime() [2934]
                0.00    0.00      20/36          CommunicationBarrier() [2972]
-----------------------------------------------

                    .                       .                       .
                    .                       .                       .
                    .                       .                       .
```

The TAU Performance System

- *"TAU Performance System is a portable profiling and tracing toolkit for performance analysis of parallel programs..."*
- `http://tau.uoregon.edu`
- Full-featured, but challenging to install and use
- On Triton Resource: `load module tau`
- Compile your code with `tau_f90.sh`, `tau_cxx.sh`, `tau_cc.sh`
- Provides multiple `Makefiles` that define script behavior, e.g.
  `TAU_MAKEFILE = $(TAUHOME)/x86_64/lib/Makefile.tau-mpi-pdt`
- `pprof`: text-based profile display
- `paraprof`: Java-based analysis / visualization tool

TAU pprof text-based utility

Terminal

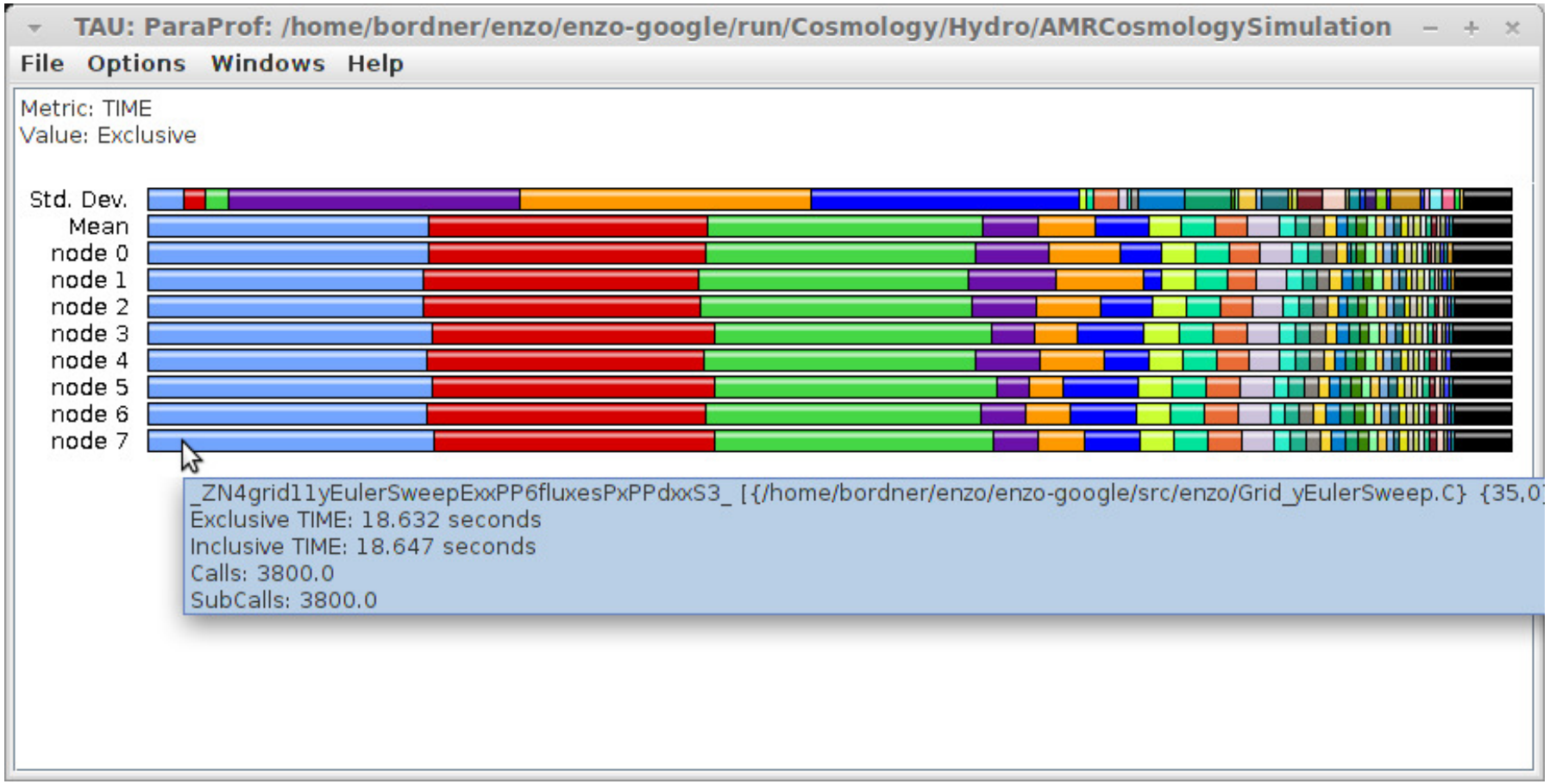
File Edit View Search Terminal Help

```
% pprof -m | head -14
Reading Profile files in profile.*

NODE 0;CONTEXT 0;THREAD 0:
---------------------------------------------------------------------------------------
%Time    Exclusive    Inclusive       #Call      #Subrs  Inclusive Name
              msec   total msec                          usec/call
---------------------------------------------------------------------------------------
 43.1        8,084        8,084         100         100      80846 long_int grid::xEulerSweep()
 42.1        7,873        7,895       20600       20600        383 long_int grid::yEulerSweep()
  3.0          359          553         800         800        692 double grid::ComputeTimeStep()
  1.0          193          193         200         200        967 long_int grid::ComputePressure()
  1.1          181          203         601        1741        339 long_int CommunicationReceiveHandle
r()
  0.7          130          130         400           0        327 long_int grid::CopyBaryonFieldToOld
BaryonField()
 99.6           49       18,670           1        2723   18670275 long_int EvolveHierarchy()
%
```

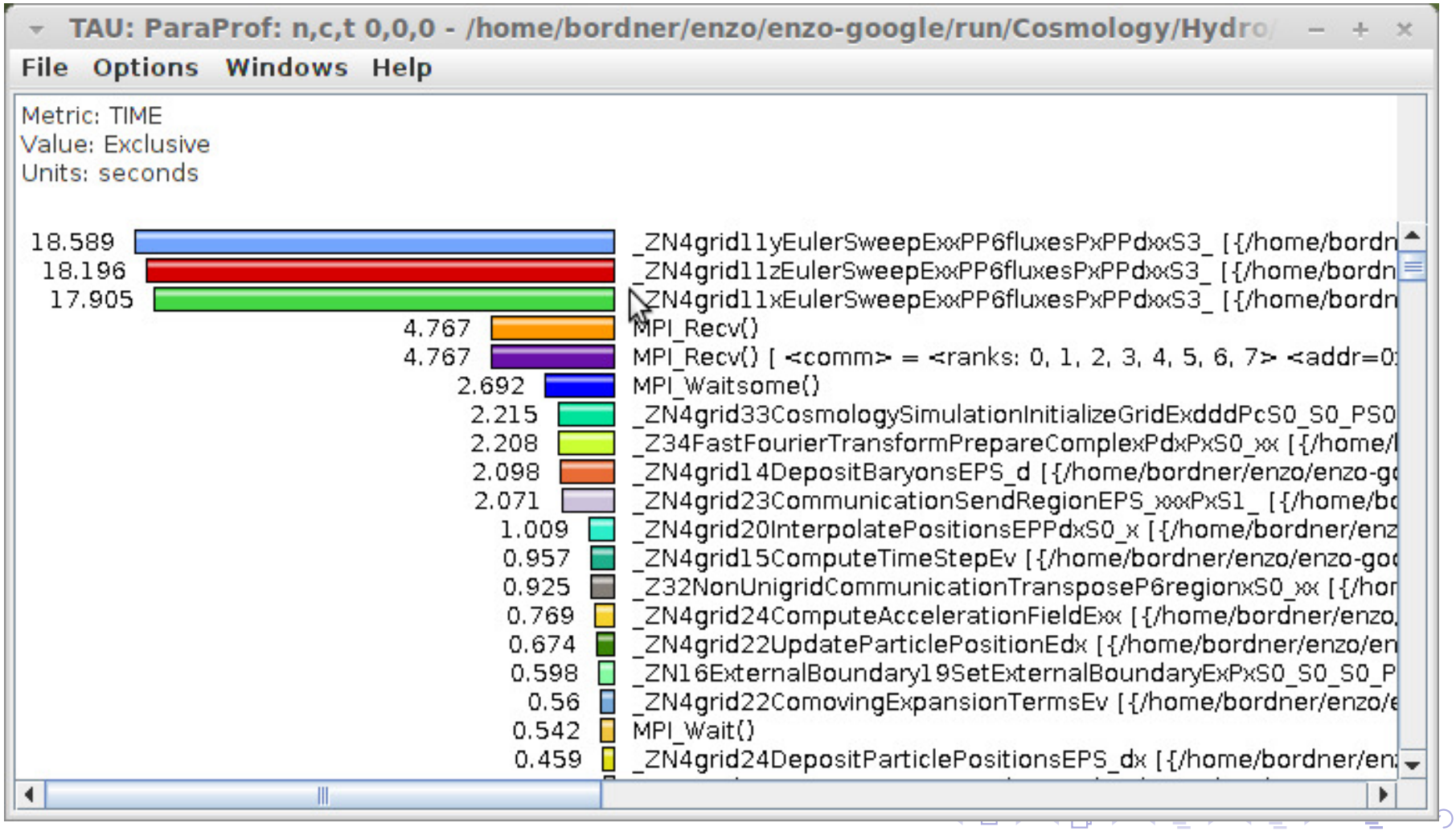
TAU `paraprof` visualization tool

Time per function on all MPI ranks



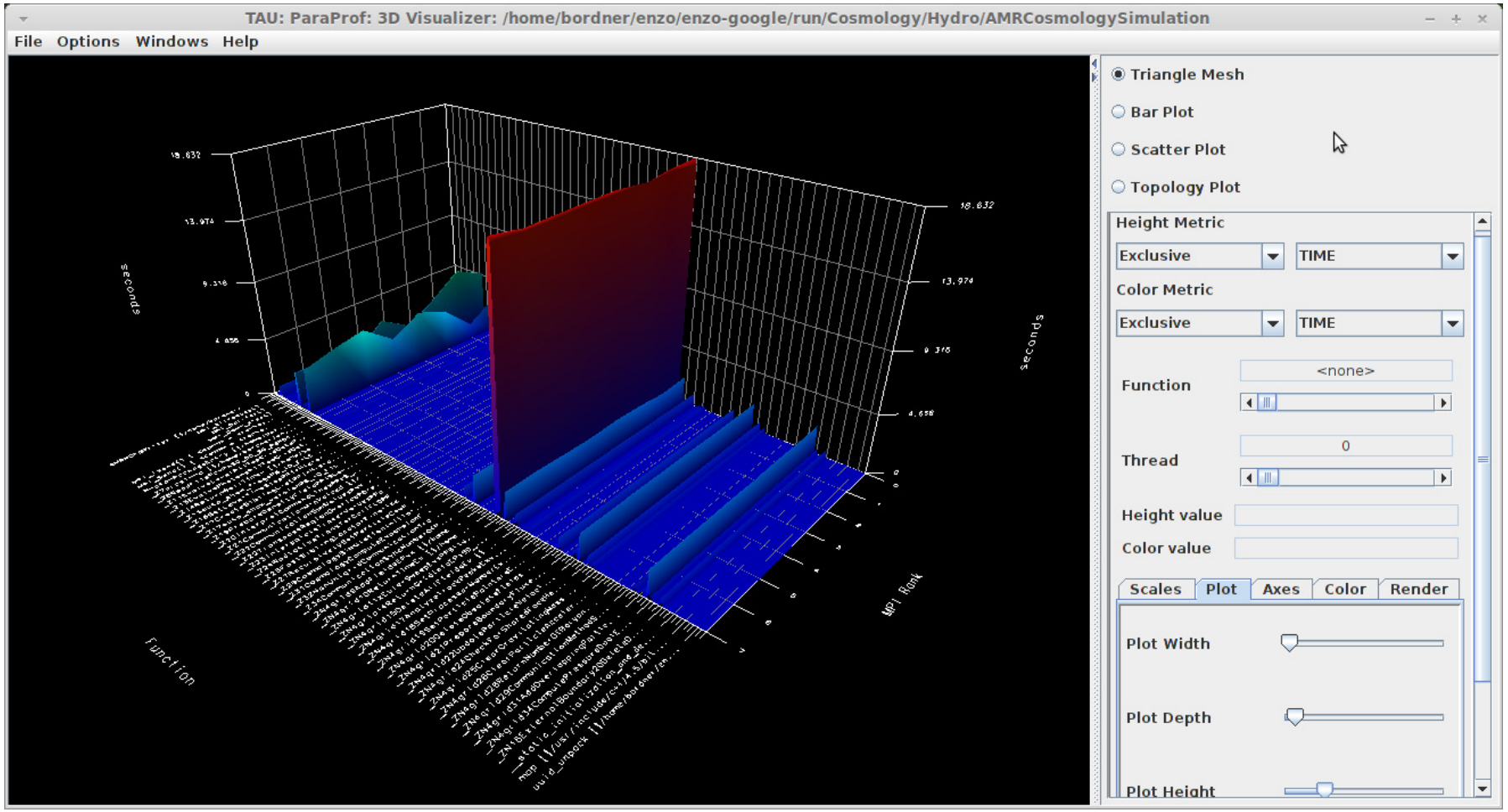
TAU paraprof visualization tool

Time per function on a single MPI rank



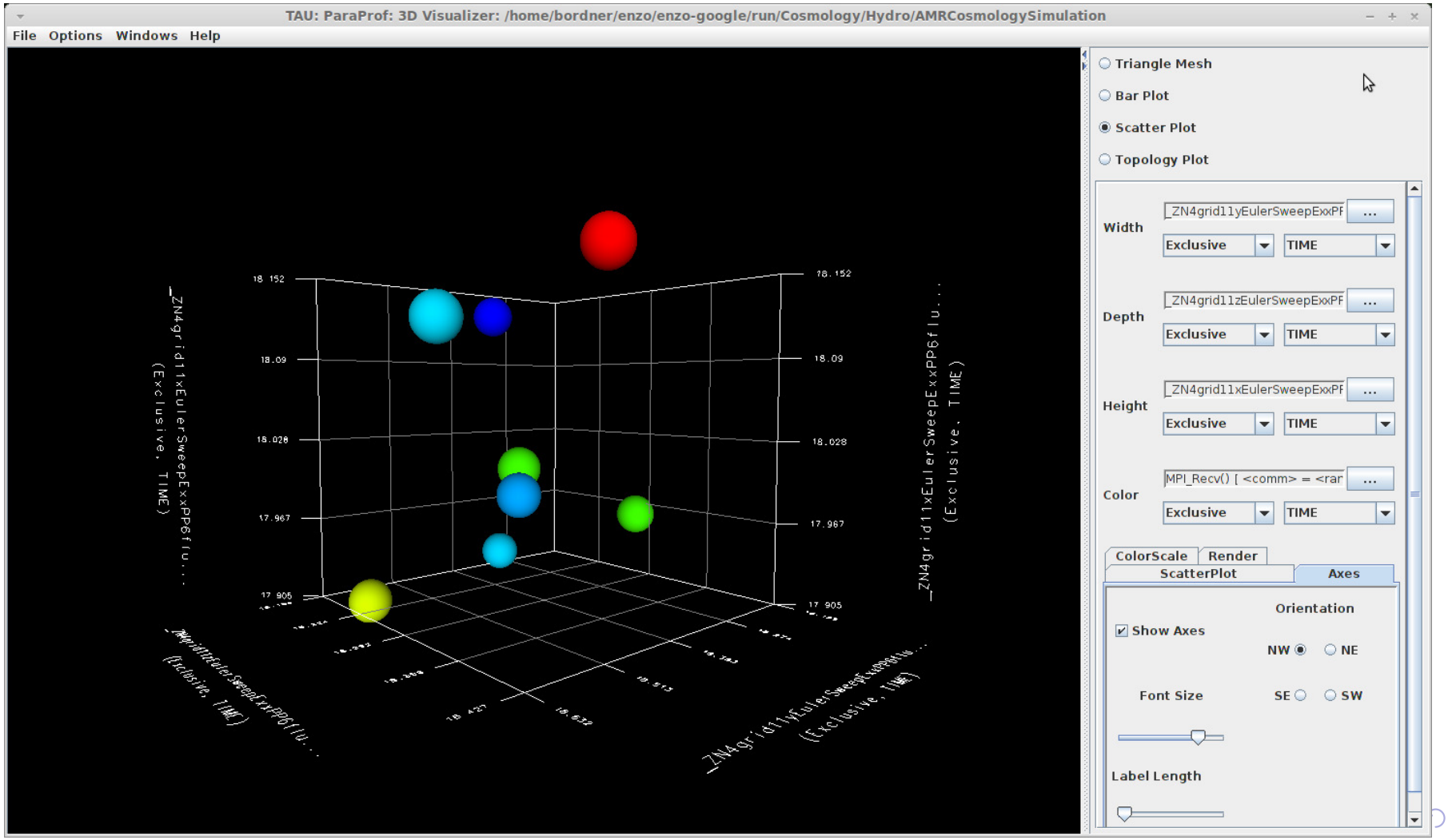
TAU paraprof visualization tool

Time per function on all MPI ranks (3D)



TAU paraprof visualization tool

Three-way scatter plot of three related functions (3D)



Profiling Hardware Performance Counters

Tools exist for accessing hardware performance counters

- Include cycles, instructions, flops, memory accesses, etc.
- Depending on the tool, instrumenting code may be necessary
- More useful information than just time
- Helpful in diagnosing CPU and memory performance problems
- `PAPI` is a software library for accessing hardware counters
- `PerfSuite` includes utilities for automatic profiling

PAPI Performance API

- PAPI specifies a standard API for accessing hardware performance counters
- http://icl.cs.utk.edu/papi/
- Can call PAPI functions directly
  - e.g. `PAPI_flops()`
  - link with PAPI library `-lpapi`
- Typically used indirectly via other tools that call PAPI

PAPI hardware performance counters

- Some typical PAPI counters
  - PAPI_TOT_CYC Total cycles
  - PAPI_TOT_INS Instructions completed
  - PAPI_FP_OPS Floating point operations
  - PAPI_LST_INS Load/store instructions completed
  - PAPI_VEC_SP Single precision vector/SIMD instructions
- Typically $>$ 50 counters available...
- But can only count a few at a time: multiplexing
- papi_avail lists and describes available counters

PerfSuite toolset

- *"PerfSuite is a collection of tools, utilities, and libraries for software performance analysis"*
- `http://sourceforge.net/projects/perfsuite/`
- `psinv`
  - displays processor information and available resources
  - usage: `psinv`
- `psrun`
  - utility for collecting performance data
  - executable need not be instrumented
  - usage: `mpirun -np <procs> psrun a.out`
- `psprocess`:
  - generates readable report from performance data file
  - usage: `psprocess a.out.6236.gedeckt.txt`

PerfSuite `psinv` utility

System and processor information

```
System Information -
Node Name:              gedeckt
OS Name:                Linux
OS Release:             2.6.38-15-generic
OS Build/Version:       #59-Ubuntu SMP Fri Apr 27 16:03:32 UTC 2012
OS Machine:             x86_64
Processors:             8
Total Memory (MB):      10013.06
System Page Size (KB):  4.00

Processor Information -
Vendor:                 Intel
Processor family:       Pentium Pro (P6)
Brand:                  Intel(R) Core(TM) i7 CPU 920 @ 2.67GHz
Model (Type):           (unknown)
Revision:               5
Clock Speed:            1733.00 MHz
```

Perfsuite `psinv` utility

Cache and TLB information

Cache and TLB Information -
Cache levels: 3

Cache Details -
Level 1:
Type: Instruction
Size: 32 KB
Line size: 64 bytes
Associativity: 4-way set associative

Type: Data
Size: 32 KB
Line size: 64 bytes
Associativity: 8-way set associative

Level 2:
Type: Unified
Size: 256 KB
Line size: 64 bytes
Associativity: 8-way set associative

.
.
.

TLB Details -
Level 1:
Type: Instruction
Entries: 64
Pagesize (KB): 4
Associativity: 4-way set associative

Type: Instruction
Entries: 7
Pagesize (KB): 2048 4096
Associativity: Fully associative

Type: Data
Entries: 64
Pagesize (KB): 4
Associativity: 4-way set associative

Type: Data
Entries: 32
Pagesize (KB): 2048 4096
Associativity: 4-way set associative

.
.
.

PerfSuite psprocess utility

Hardware counters from an MPI run with psrun

```
 1 Conditional branch instructions.....  1,597,794,454
 2 Branch instructions.................  2,010,337,100
 3 Conditional branch instructions mispredicted.......     45,215,254
 4 Conditional branch instructions taken..............  1,359,594,484
 5 Floating point operations...........  1,603,445,112
 6 Level 1 data cache accesses........  6,016,074,767
 7 Level 1 data cache misses...........    125,896,044
 8 Level 1 instruction cache accesses.  5,387,031,022
 9 Level 1 instruction cache misses....     44,345,992
10 Level 2 instruction cache accesses..     36,546,338
11 Level 2 instruction cache misses....     11,850,854
12 Level 2 total cache accesses........    235,426,115
13 Level 2 cache misses................     60,177,610
14 Load instructions...................  5,398,862,506
15 Cycles stalled on any resource......  5,649,682,373
16 Store instructions..................  2,110,905,697
17 Data translation lookaside buffer misses..........      2,820,165
18 Instruction translation lookaside buffer misses....        979,828
19 Total cycles.......................  19,178,291,793
20 Instructions issued................  13,163,918,870
21 Instructions completed.............  15,815,031,288
```

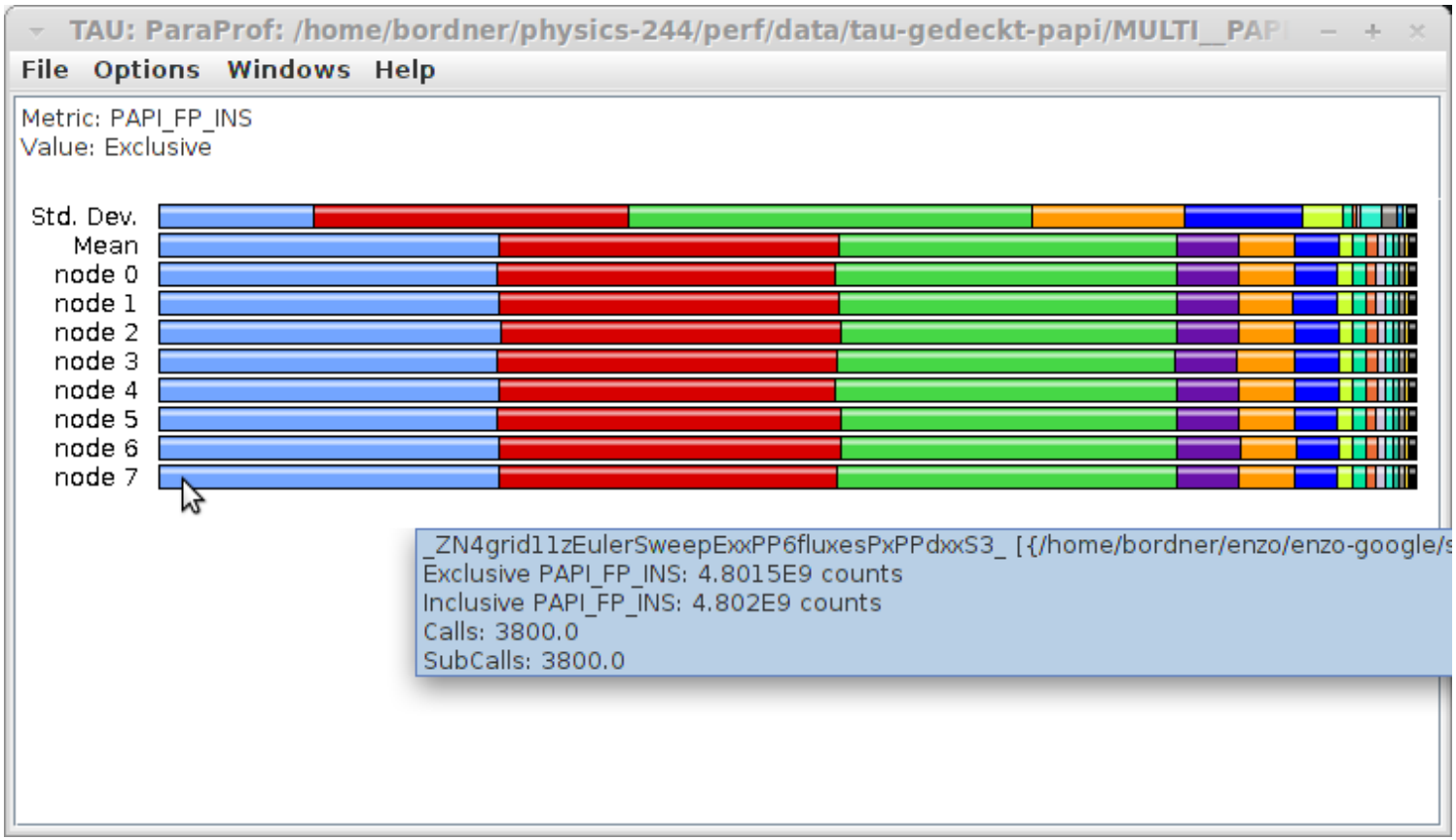
PerfSuite psprocess utility

Derived metrics from an MPI run with psrun

```
Floating point operations per cycle.....................      0.084
Floating point operations per graduated instruction.....      0.101
Graduated instructions per cycle........................      0.825
Issued instructions per cycle...........................      0.686
Graduated instructions per issued instruction...........      1.201
Issued instructions per level 1 instruction cache miss......  296.846
Graduated instructions per level 1 instruction cache miss.....  356.628
Level 1 data cache accesses per graduated instruction.....      0.380
% cycles stalled on any resource........................     29.459
Graduated loads and stores per floating point operation.....      4.684
Level 1 instruction cache misses per issued instruction.....      0.003
Level 1 cache miss ratio (data).........................      0.021
Level 1 cache miss ratio (instruction)..................      0.008
Level 2 cache miss ratio (data), data cache miss and access counts    0.243
Level 2 cache miss ratio (instruction)..................      0.324
Bandwidth used to level 2 cache (MB/s).....................  321.311
MFLOPS (cycles).............................................  133.772
MFLOPS (wall clock).........................................  136.392
MIPS (cycles)...............................................  1,319.411
MIPS (wall clock)...........................................  1,345.254
CPU time (seconds)..........................................   11.986
Wall clock time (seconds)...................................   11.756
% CPU utilization...........................................  101.959
```

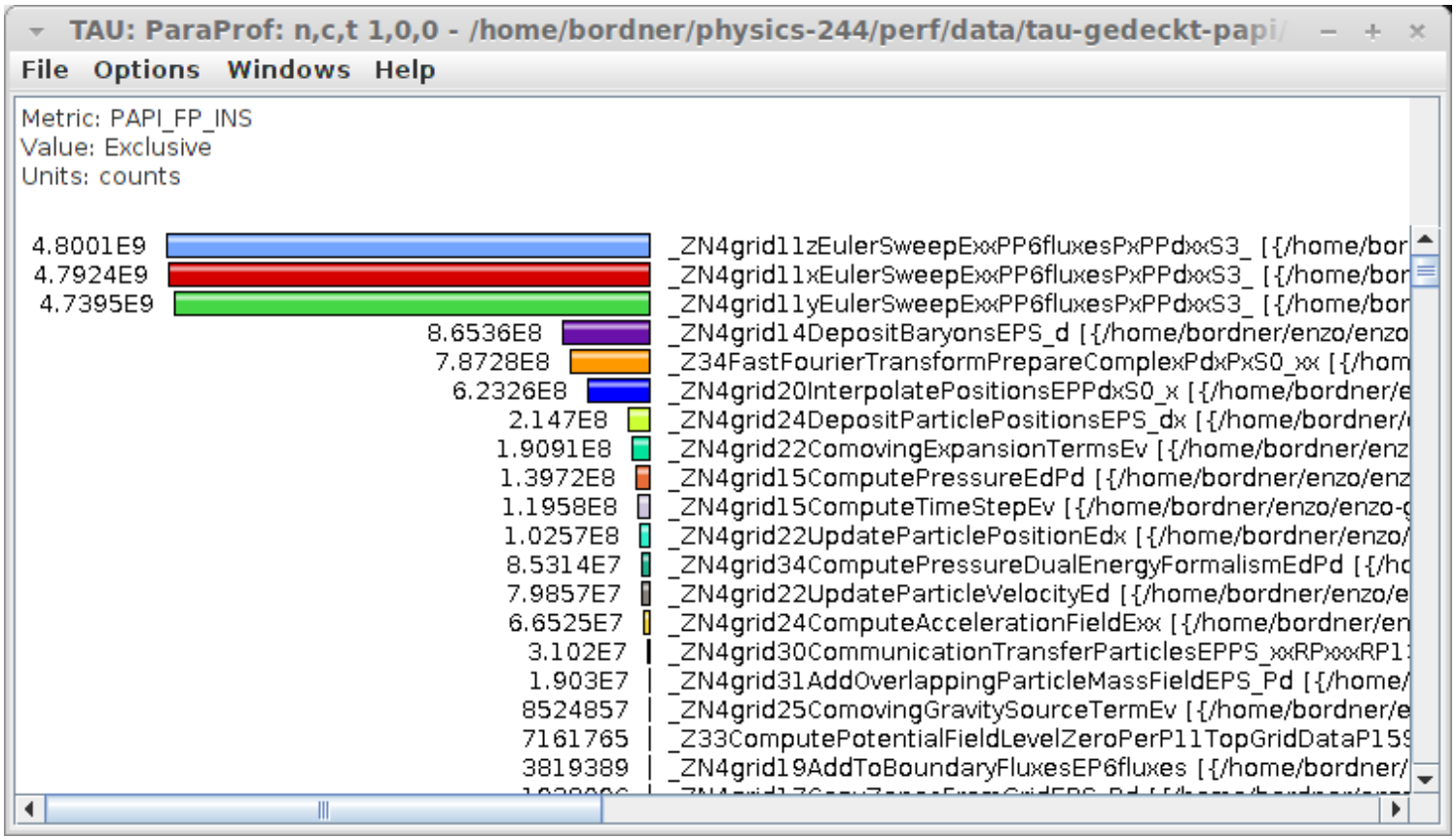
TAU `paraprof` visualization tool

Floating point operations per function on all MPI ranks



TAU paraprof visualization tool

Floating point operations on a single MPI rank



The IPM MPI profiler

- *"IPM is a portable profiling infrastructure for parallel codes."*
- http://ipm-hpc.sourceforge.net/
- IPM is easy to use: link code with -lipm
- Collects a variety of performance data:
  timing + hardware counters + MPI communication
- Outputs both text and optionally a web page containing tables, plots, graphs
- Use ipm_parse to get an automatically-generated web page:
  - setenv IPM_KEYFILE $(IPM_PATH)/ipm/ipm_key
  - ipm_parse -html bordner.1305495025.269849.0
  - http://client65-77.sdsc.edu/~bordner/ipm/

IPM text output

Top-level performance summary

```
##IPMv0.983##############################################################################
#
# command : enzo.exe AMRCosmologySimulation.enzo  (completed)
# host    : tcc-2-52/x86_64_Linux          mpi_tasks : 64 on 1 nodes
# start   : 05/30/12/23:47:40              wallclock : 180.767659 sec
# stop    : 05/30/12/23:50:41              %comm     : 74.28
# gbytes  : 0.00000e+00 total              gflop/sec : 0.00000e+00 total
#
##############################################################################
```

IPM text output

Global timing statistics

```
##############################################################################
# region  : *       [ntasks] =     64
#
#                           [total]         <avg>           min           max
# entries                        64             1             1             1
# wallclock                 11566.1        180.72       180.666       180.768
# user                      1125.87       17.5917       16.3725       18.7122
# system                    238.445        3.7257       2.44563       6.13407
# mpi                       8593.49       134.273       129.444       140.434
# %comm                                   74.2795        71.646       77.7309
##############################################################################
```

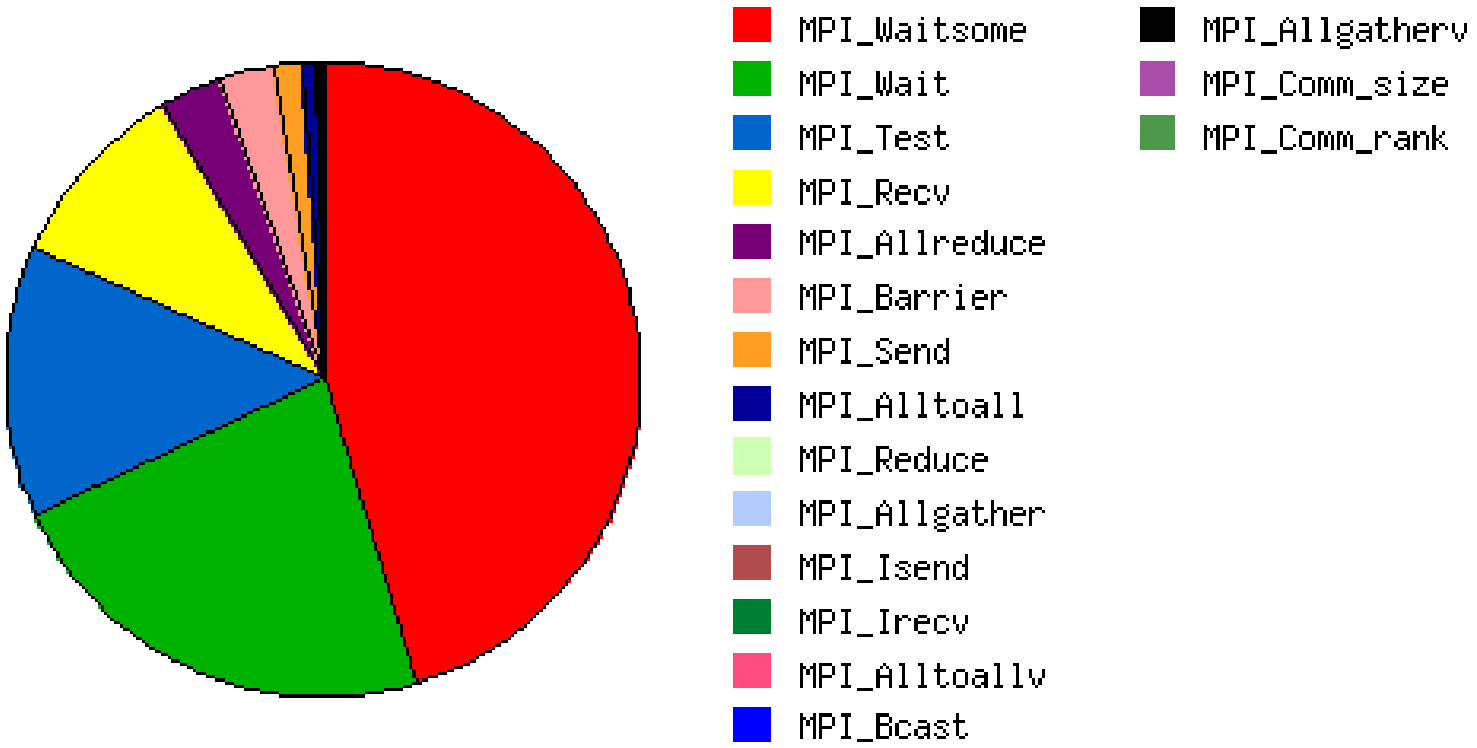
IPM text output

MPI performance

```
##########################################################################
#                            [time]       [calls]        <%mpi>      <%wall>
# MPI_Waitsome              3888.43        553887         45.25        33.62
# MPI_Wait                  1958.19     2.4192e+06         22.79        16.93
# MPI_Test                  1182.09    1.15384e+06         13.76        10.22
# MPI_Recv                   834.81        166400          9.71         7.22
# MPI_Allreduce             282.918         94336          3.29         2.45
# MPI_Barrier               237.251         14464          2.76         2.05
# MPI_Send                  120.854     2.4192e+06          1.41         1.04
# MPI_Alltoall              52.0787         25856          0.61         0.45
# MPI_Reduce                17.0491         25856          0.20         0.15
# MPI_Allgather             14.1805         12864          0.17         0.12
# MPI_Isend                 2.06106        833664          0.02         0.02
# MPI_Irecv                 2.03514    3.08646e+06          0.02         0.02
# MPI_Alltoallv              1.0919         25856          0.01         0.01
##########################################################################
```

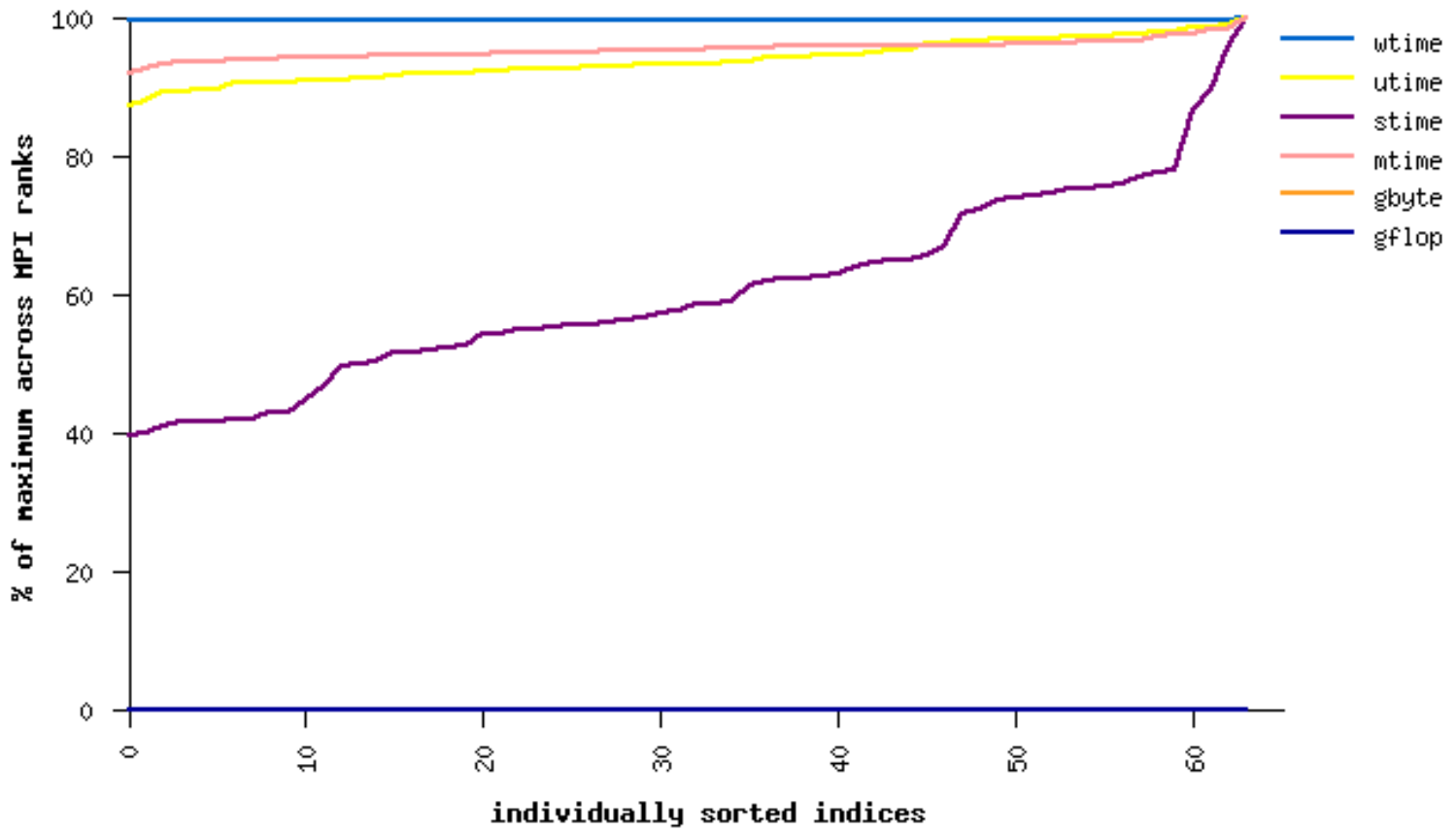
IPM web page output

Time spent in MPI functions



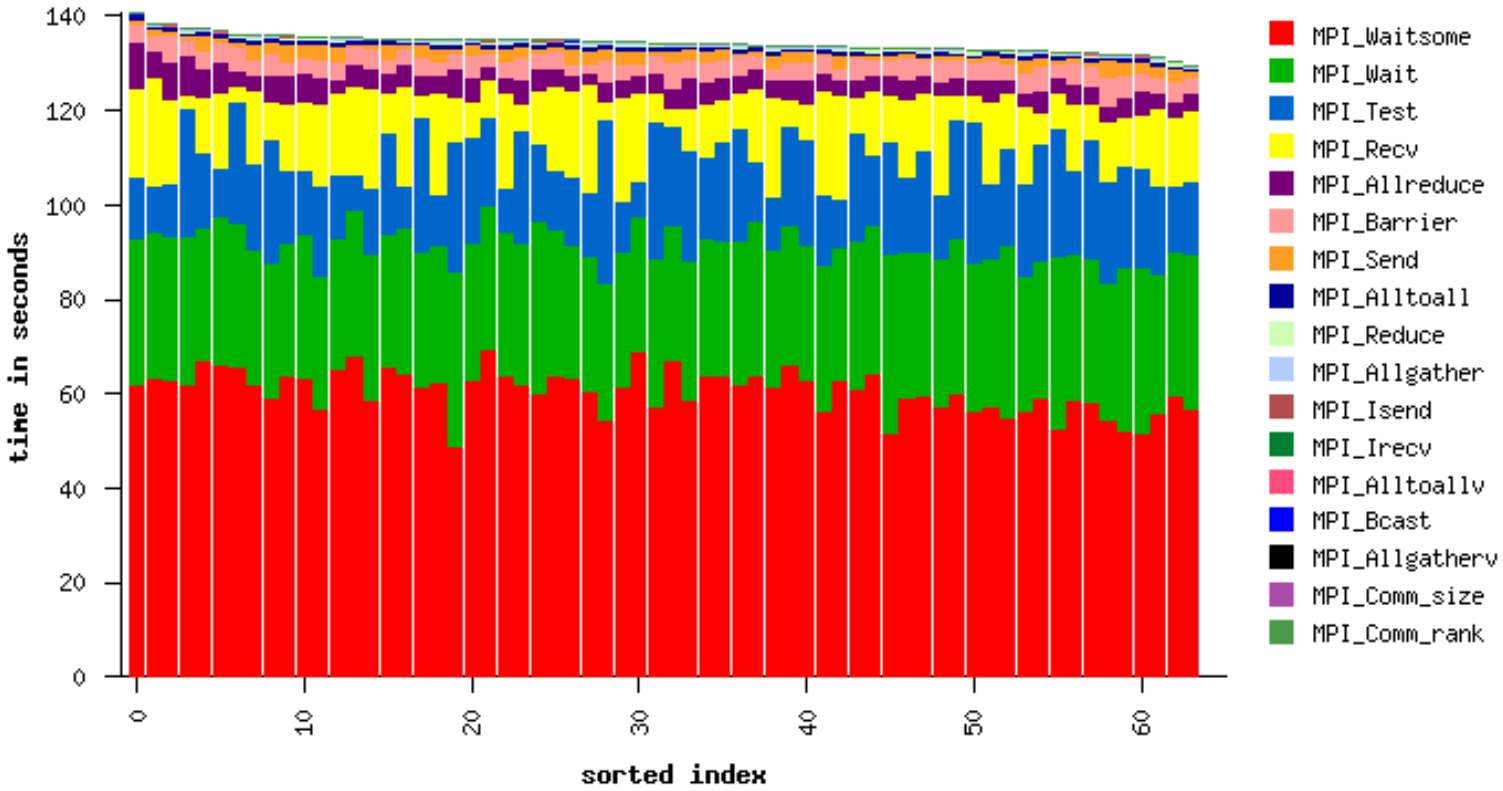
IPM web page output

Wall, user, system, and MPI time per MPI rank



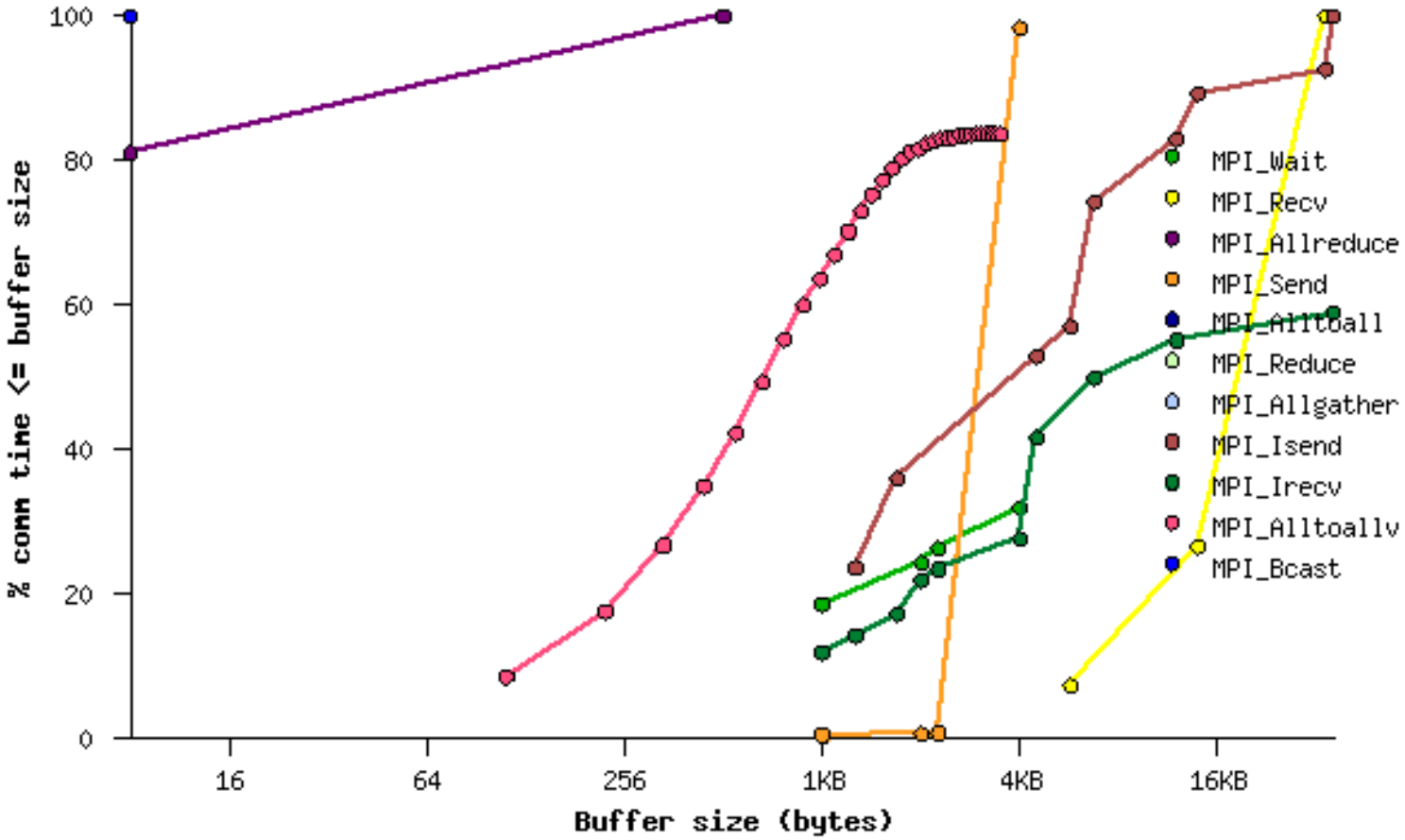
IPM web page output

Time spent in MPI functions for each MPI rank



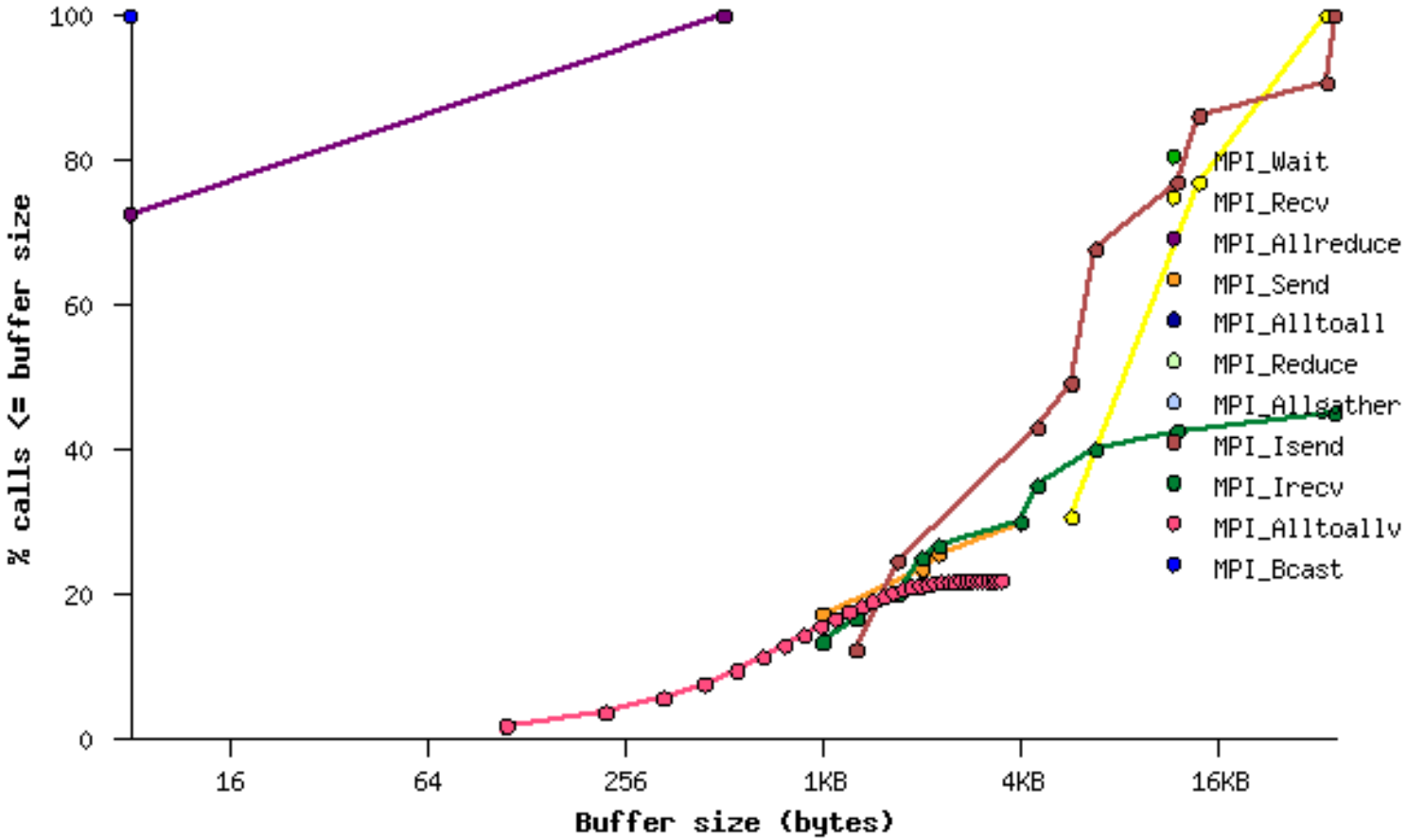
IPM web page output

Time spent in MPI functions by buffer size



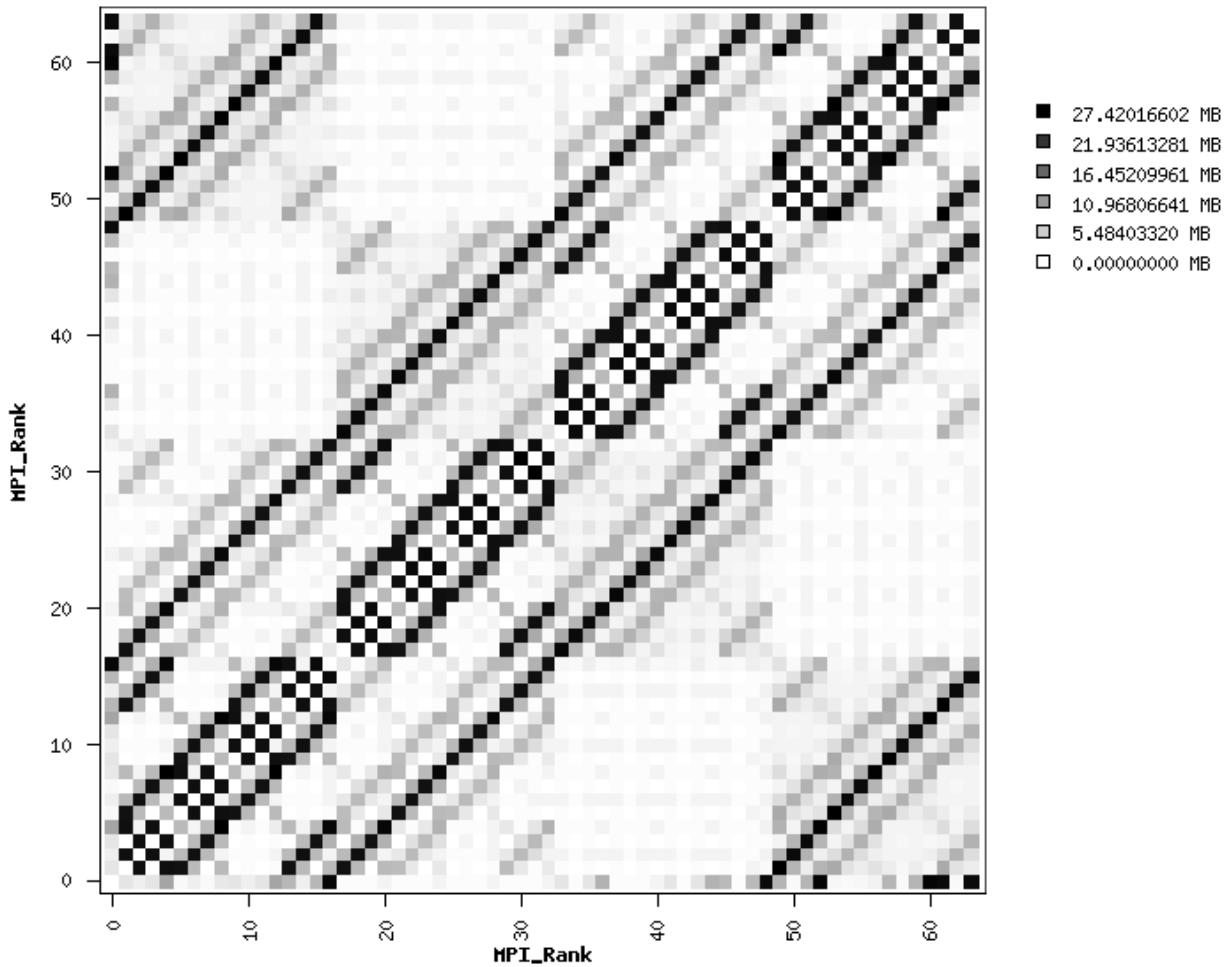
IPM web page output

Number of MPI function calls by buffer size



IPM web page output

Total communication between each pair of MPI ranks



Tracing MPI

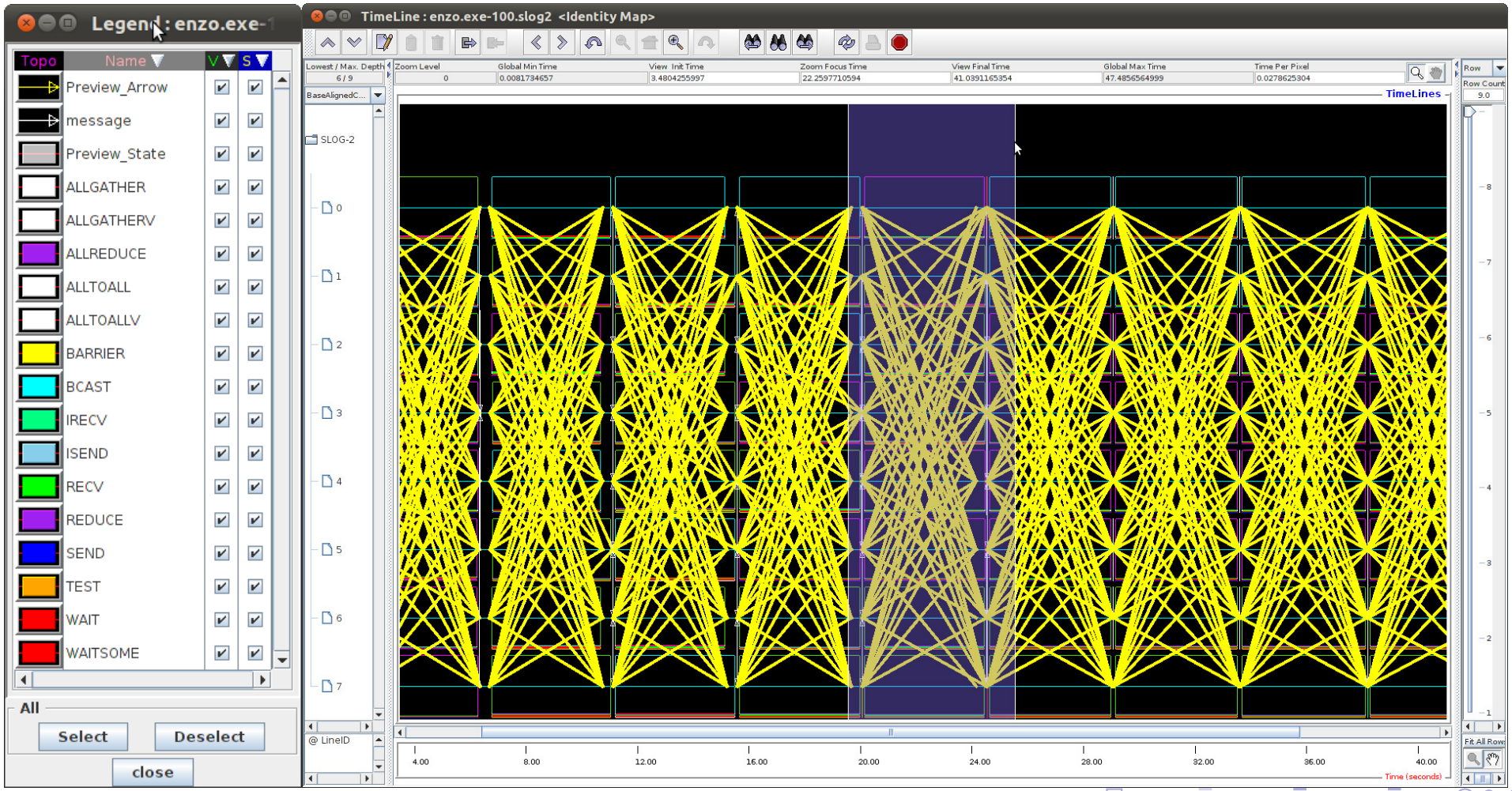
- MPI tracing tools trace every call to MPI
- Typically store begin/end times, message size, MPI rank(s)
- Useful for investigating dynamic communication patterns
- Log files can be huge
- `MPE` for logging MPI trace data
- `Jumpshot` for visualizing trace files

MPE / Jumpshot

- *"A set of profiling libraries to collect information about the behavior of MPI programs"*
- `http://www.mcs.anl.gov/research/projects/perfvis/`
- Easy to use: link code with `-llmpe -lmpe`
- Additional functions for controlling MPI tracing
- Utilities for converting between trace file formats
- Includes `Jumpshot` for visualizing MPE trace files

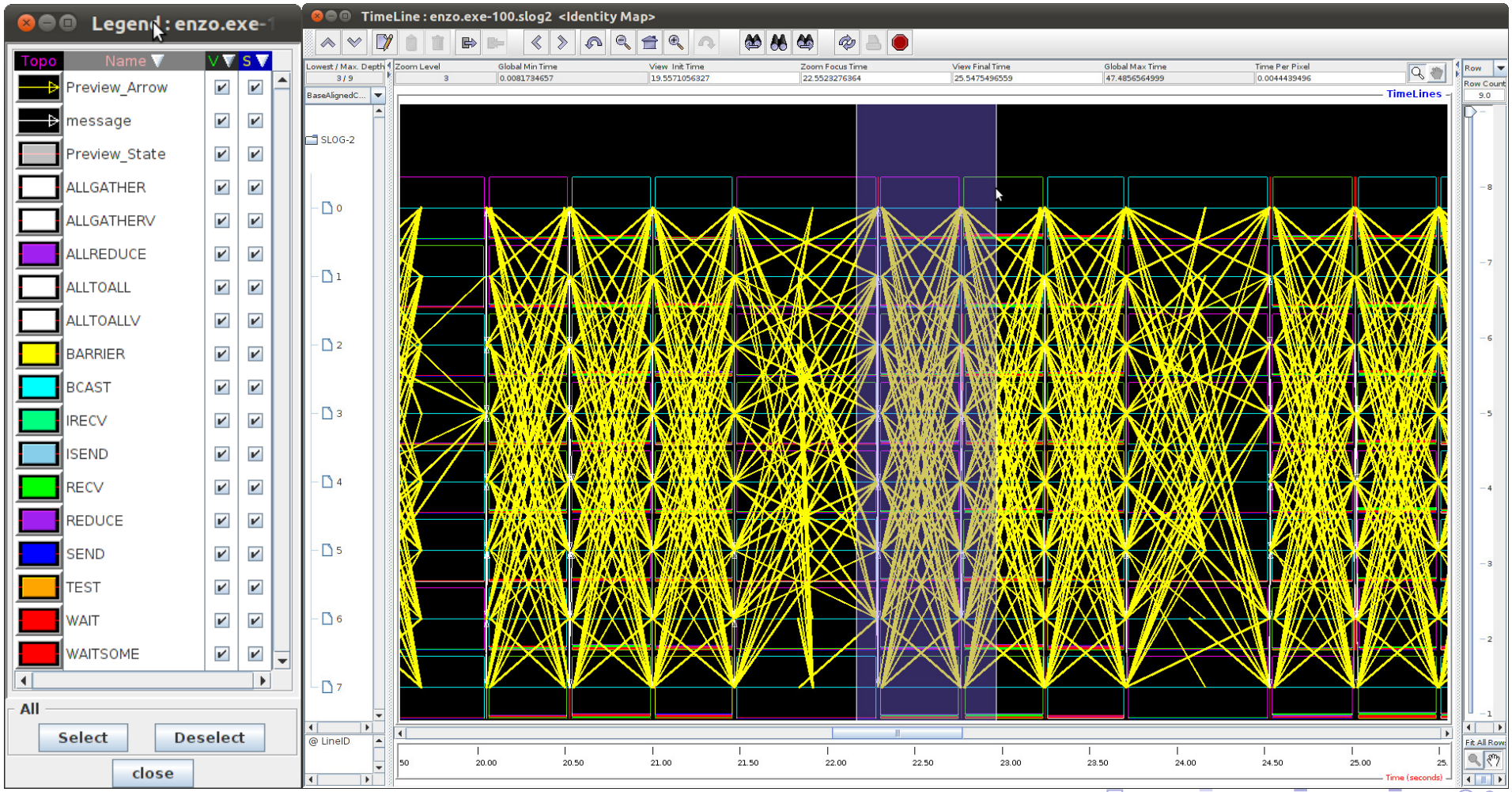
Jumpshot MPI visualization

Entire run: 4s to 40s



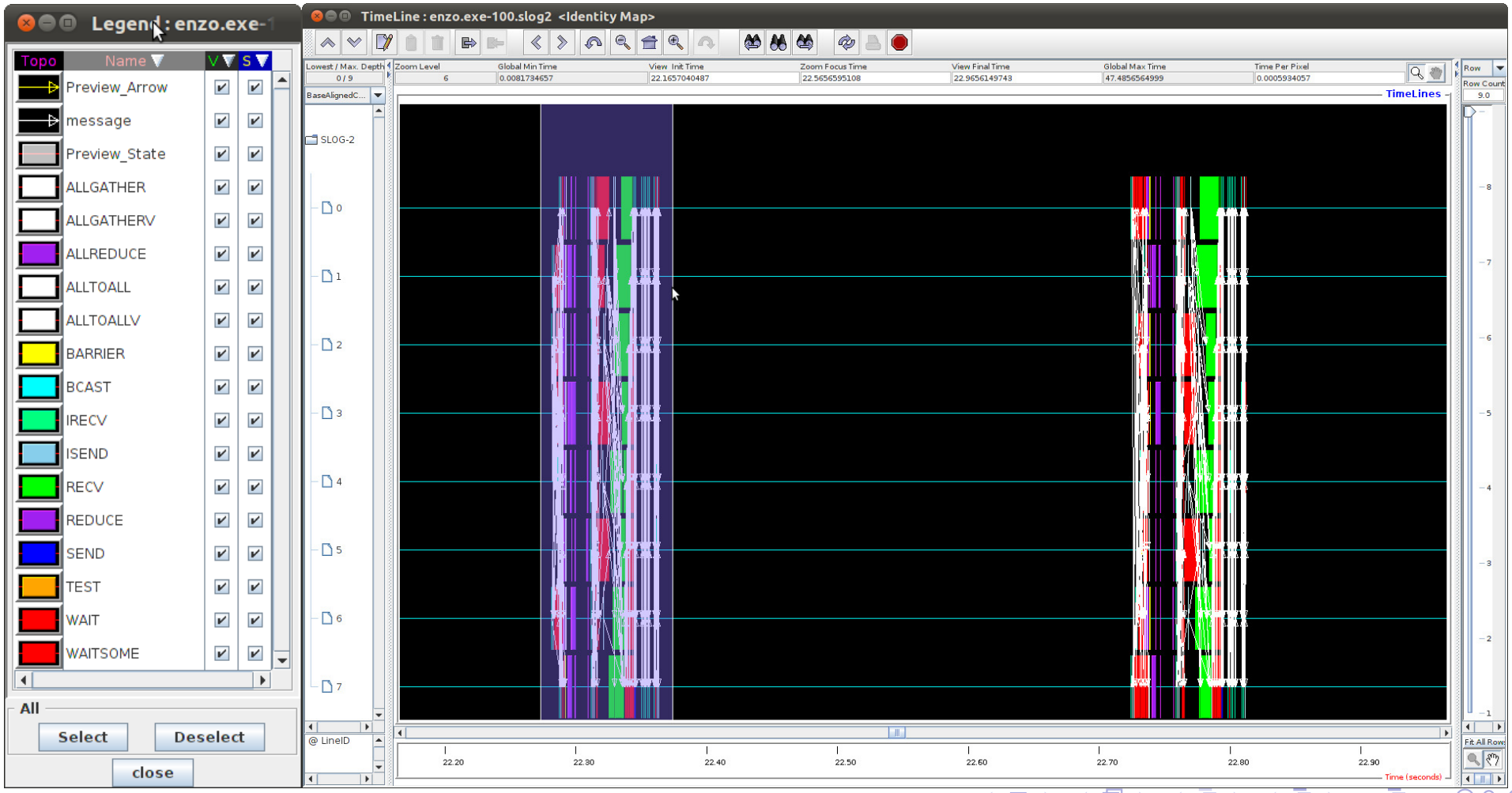
Jumpshot MPI visualization

Large portion of run: 20s to 25s



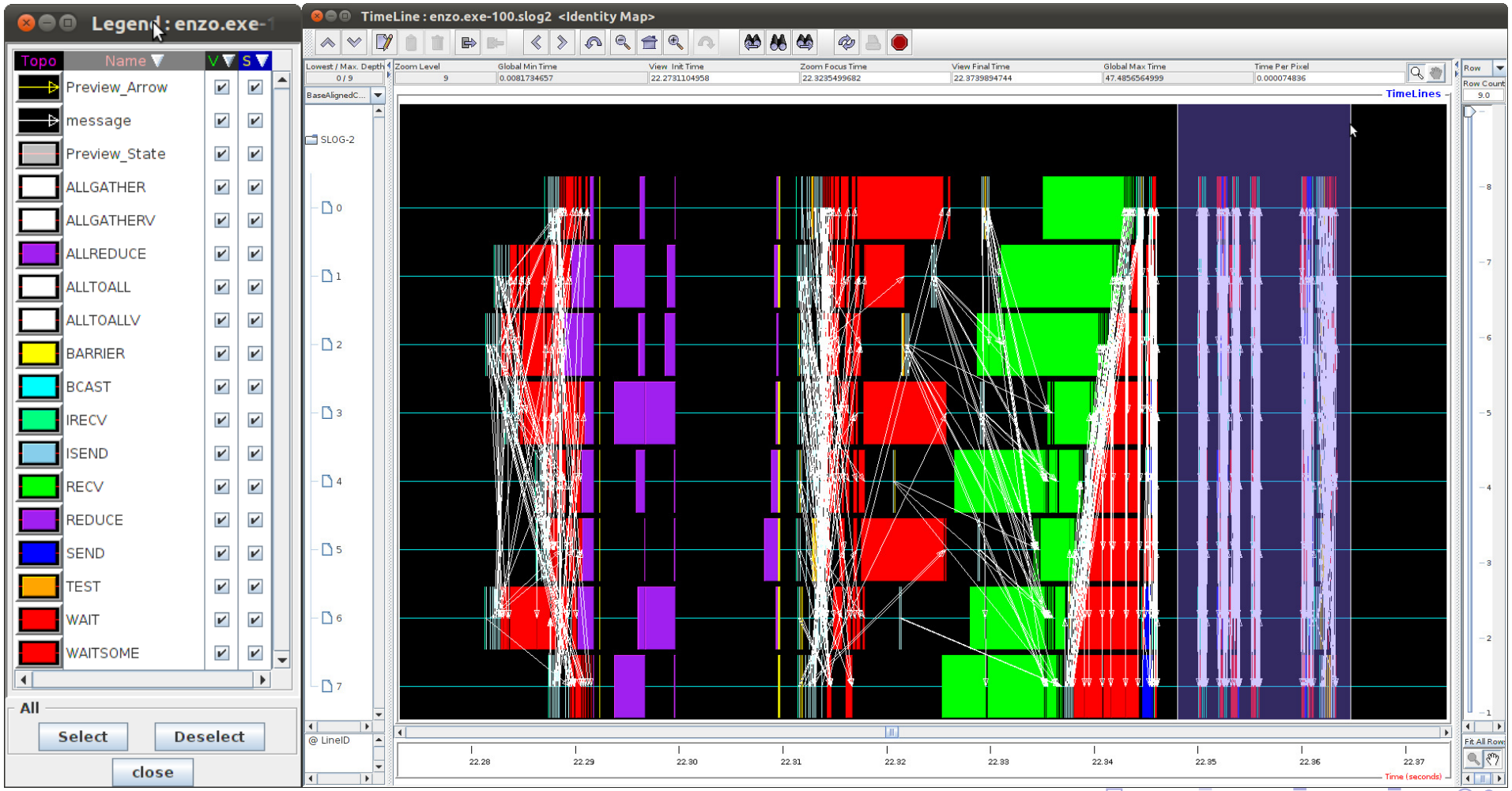
Jumpshot MPI visualization

Individual timesteps: 22.2s to 22.9s



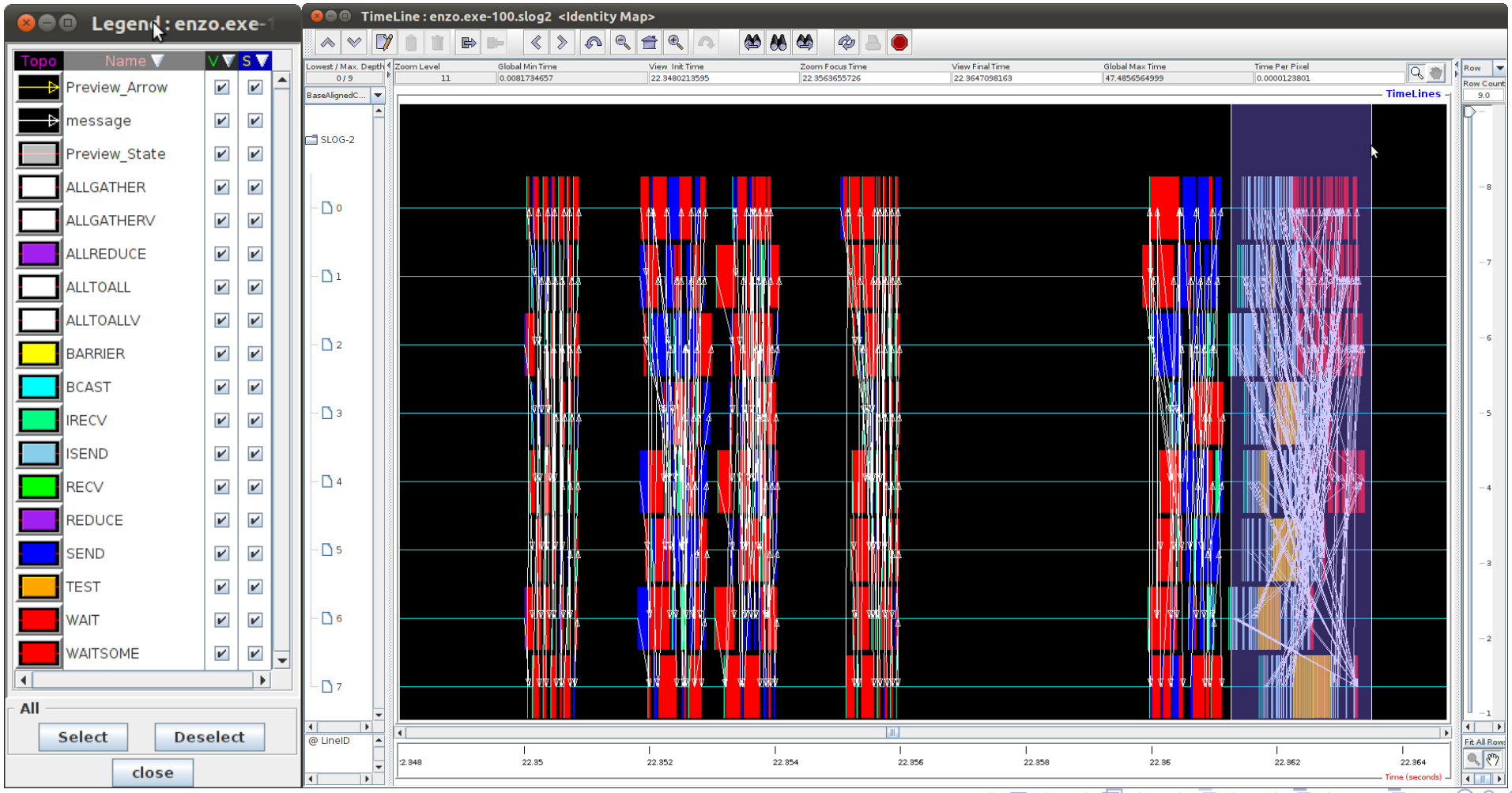
Jumpshot MPI visualization

Communication for one timestep: 22.28s to 22.37s



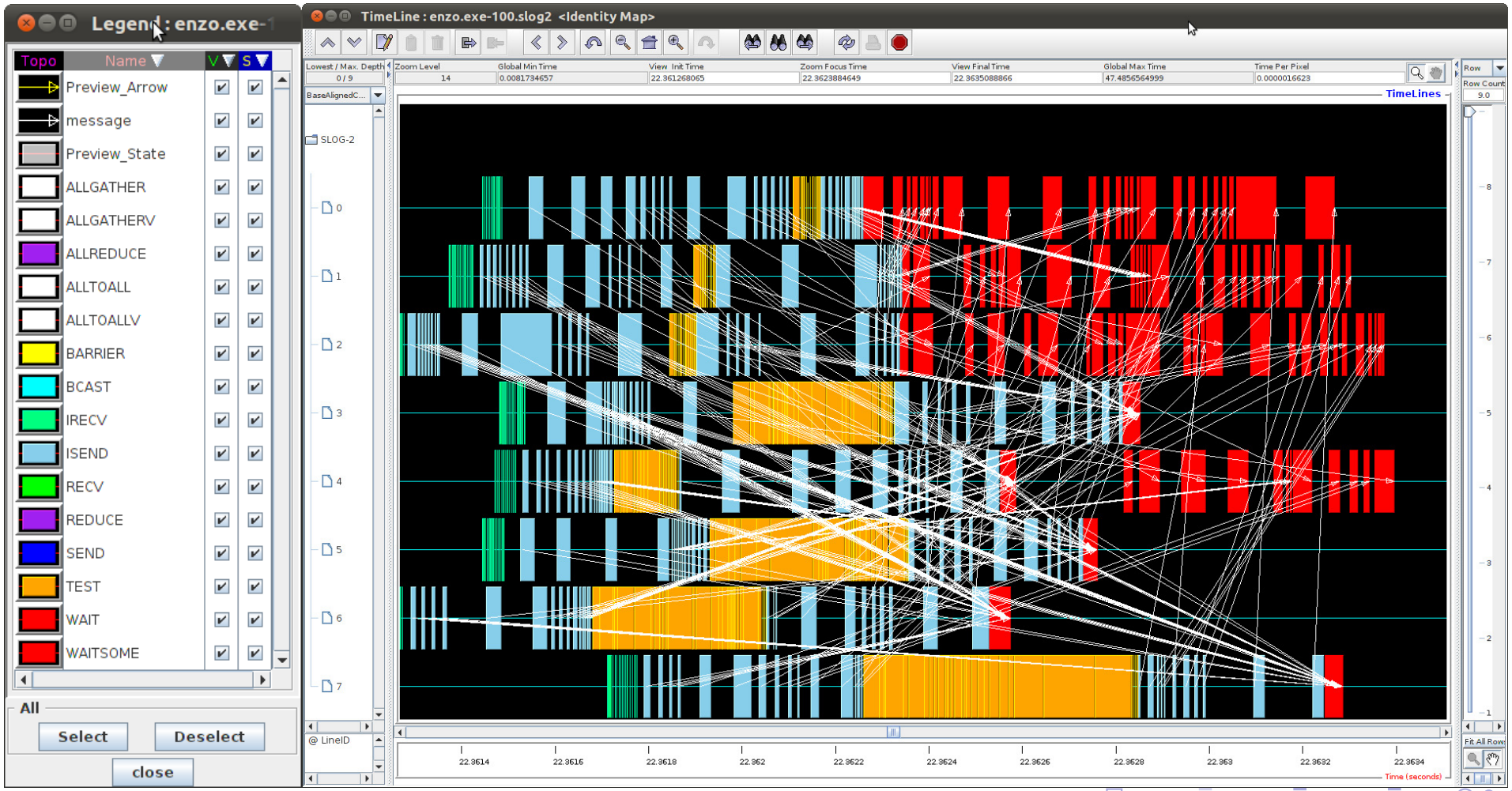
Jumpshot MPI visualization

Communication within one timestep: 22.95s to 22.964s



Jumpshot MPI visualization

Individual MPI calls: 22.9614s to 22.9634s



Conclusions

Some optimization guidelines

- Try different compilers if several available
- Experiment with compiler optimizations
  - The highest optimization level is not necessarily the fastest!
- Computation guidelines
  - Fuse loops to reduce loop overhead
  - Unroll loops to improve utilization of multiple FPU's
- Memory access guidelines
  - Access arrays using stride-1 when possible
  - Avoid multiple sweeps through large arrays
- Communication guidelines
  - Use vendor-supplied MPI libraries if available
  - Merge multiple short sends into single send if possible
  - Try non-blocking MPI / one-sided MPI-2 communication