A Comparison of Performance Analysis Tools on the NERSC SP

Jonathan Carter

NERSC User Services

Performance Tools on the IBM SP

- PE Benchmarker
  - IBM PSSP
  - Trace and visualize hardware counters values or MPI and user-defined events
- Paraver
  - European Center for Parallelism at Barcelona (CEPBA)
  - Trace and visualize program states, hardware counters values, and MPI and user-defined events
- Vampir
  - Pallas GmbH
  - Trace and visualize MPI and user-defined events
- PAPI, HPMlib, poe+, etc.
  - Tools based on pmapi hardware-counter interface
  - Summaries over entire execution

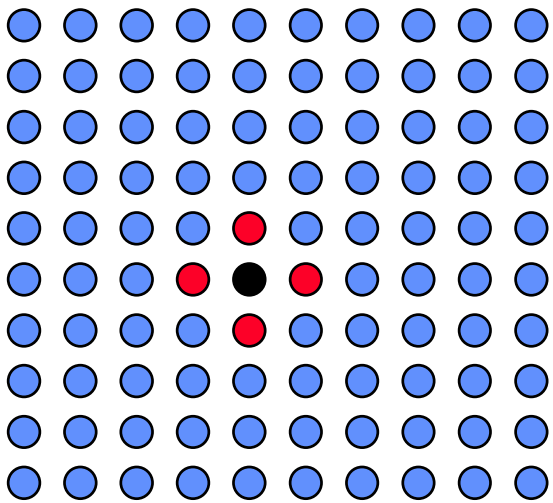
Example Program

- Solve Poisson equation on a 2D grid using Jacobi iteration

$$\nabla^2 u = f(x,y)$$

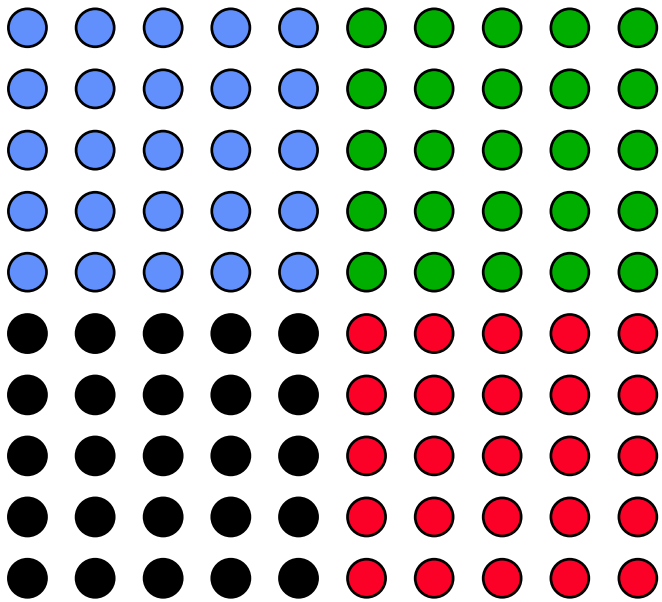
$$u(x,y) = g(x,y)$$

$$u_{i,j}^{k+1} = \tfrac{1}{4}(u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} + u_{i+1,j}^{k} - h^2 f_{i,j})$$



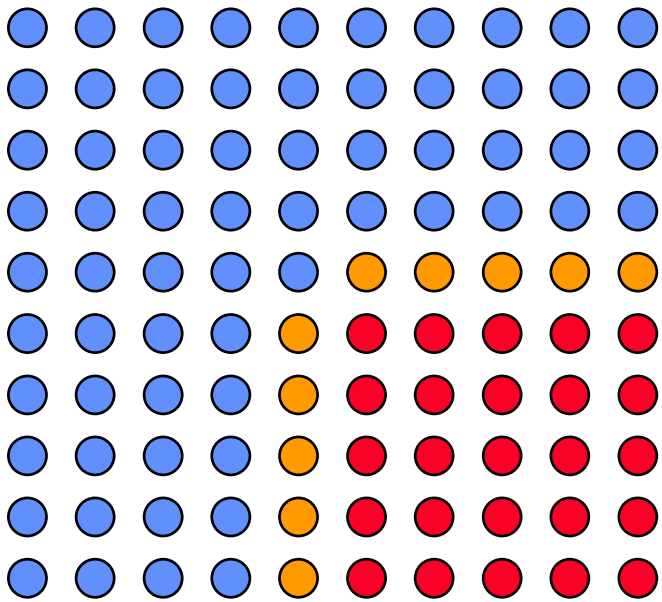
Example Program

- Decompose 2D grid into blocks, giving a block to each MPI process
- Assuming 4 MPI processes:



Example Program

- At the edges, information must be exchanged between MPI processes:



Example Program

- Flow chart of execution:

```
initialize
while ! converged
   exchange boundary data
   jacobi
   collect global sum of differences
   update
finalize
```

- Each MPI process is parallelized using OpenMP for the `jacobi` and `update` phases.

PE Benchmarker

- Performance Collection Tool (pct)
  - Application is run under the control of pct, uses DPCL to insert probes into executable before it runs.
  - Collects hardware/OS profiles at a thread level on a routine by routine basis
  - Collects MPI event statistics
- Performance Visualization Tool (pvt)
  - Graphical display of execution statistics
- MPI events
  - Summaries only (utestats), relies on ANL *jumpshot* to visualize MPI timelines

Performance Collection Tool

- Available with a graphical or command line interface
- May collect MPI event statistics or hardware/OS profiles (hardware counters, cpu time, etc.)
- Limitations:
  - Instrumentation takes place after program has been launched by POE. For a large number of source files and routines, this can take a considerable amount of time.
  - Requires 512 MB of shared memory segments

Performance Collection Tool

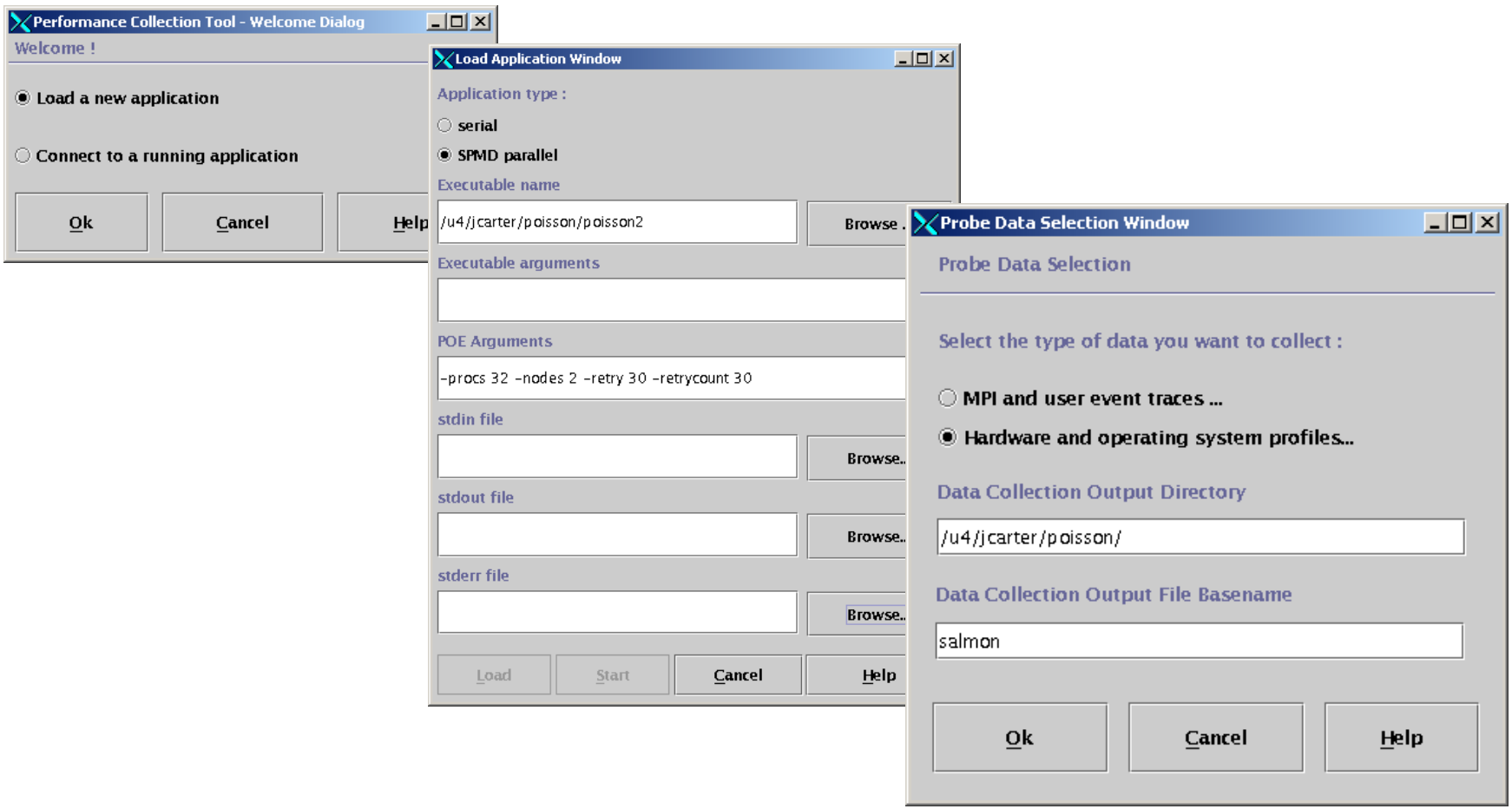
- For MPI event statistics
  - Select subset of MPI processes
  - Select subset of source files or routines
  - Select subset of MPI events
  - Add user-defined events
- For hardware/OS profiles
  - Select subset of processes
  - Select subset of timers, memory usage, hardware-counter group
  - Define hardware-counter groups (several already predefined)

Performance Collection Tool

- Must use "threaded" compilers for MPI tracing
- Must set environment variable MP_UTE=YES before linking MPI application if tracing of MPI events is required
- To run

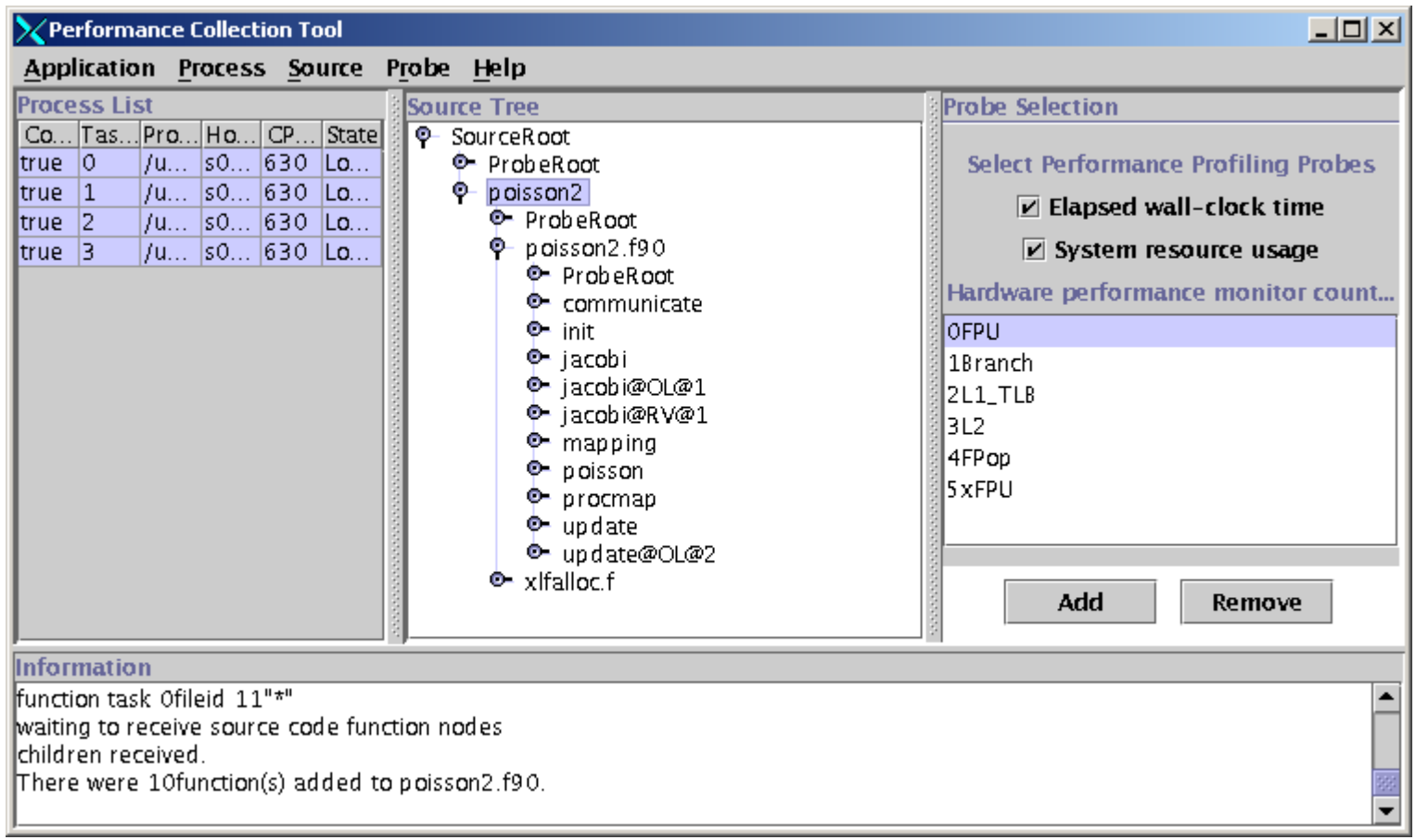
```
module load java
pct
```

Performance Collection Tool



Performance Collection Tool – Profiles

Select processes, routines, data to be collected:

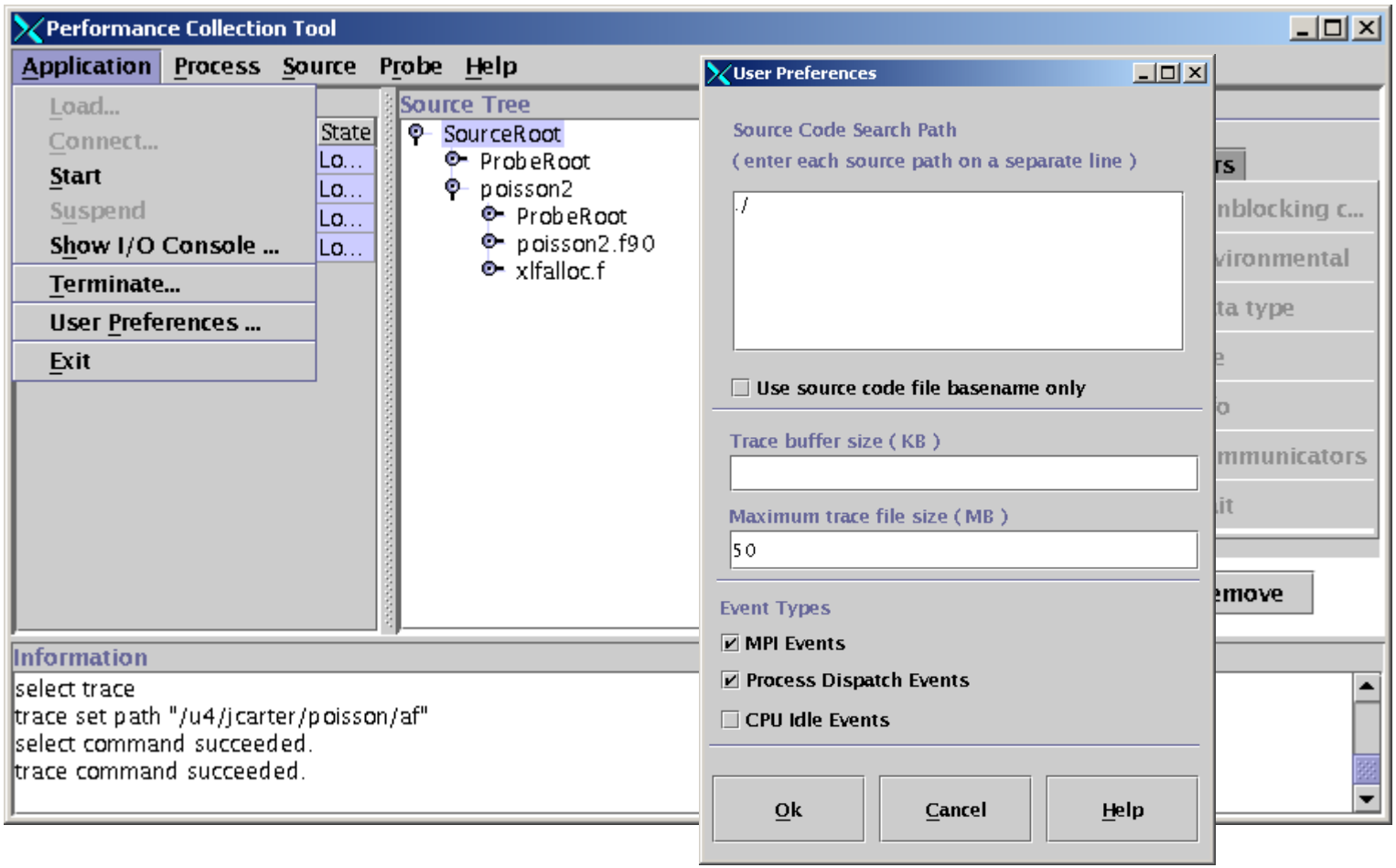


Performance Collection Tool – Profiles

- After the application completes:
  - Files named *basename*.cdf.*xx* created, one per process
  - Contain hardware profiles that can be viewed with the Performance Visualization Tool
  - Generally files are not too large

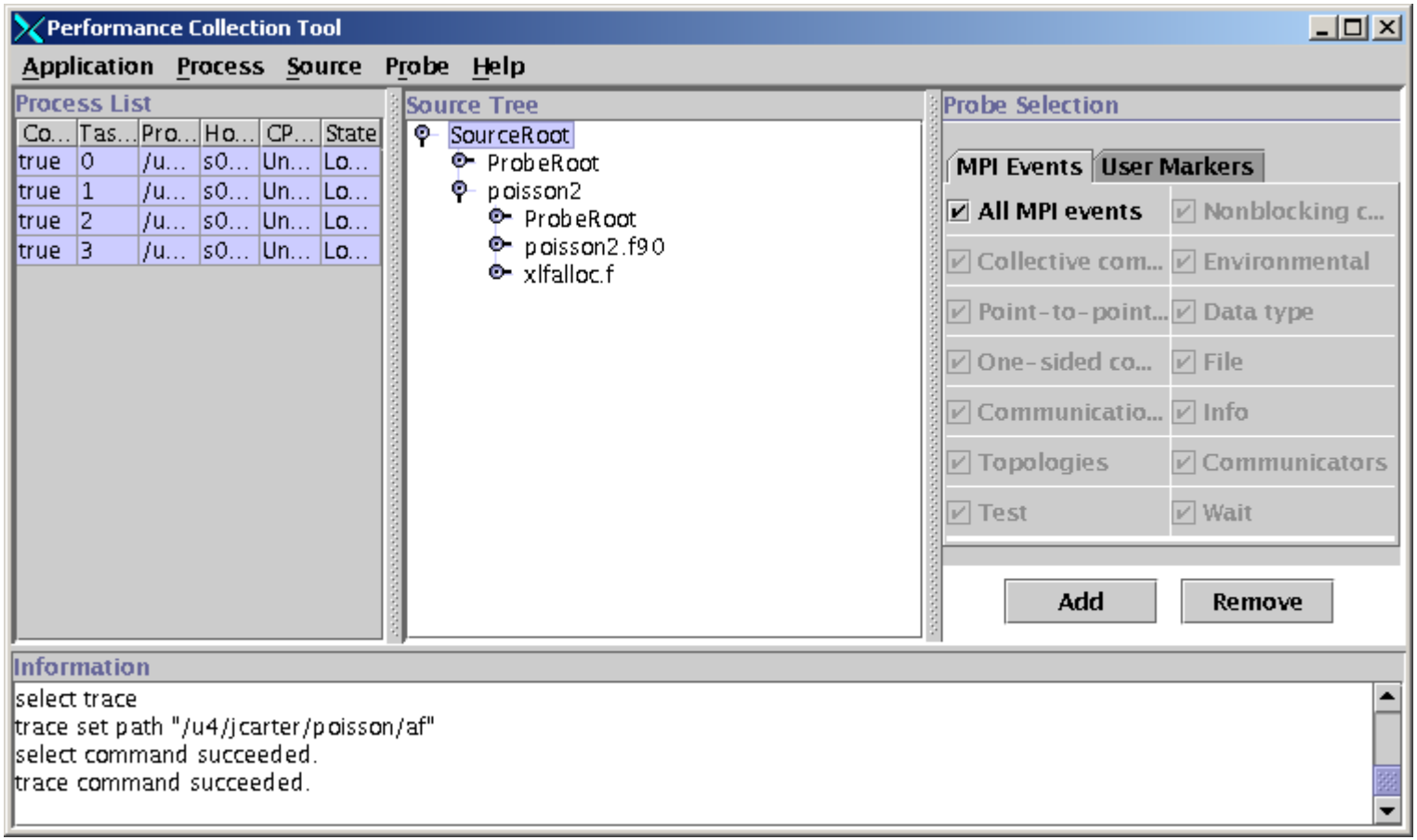
Performance Collection Tool – MPI Events

May need to increase Maximum trace file size, decide on Event Types to be monitored:



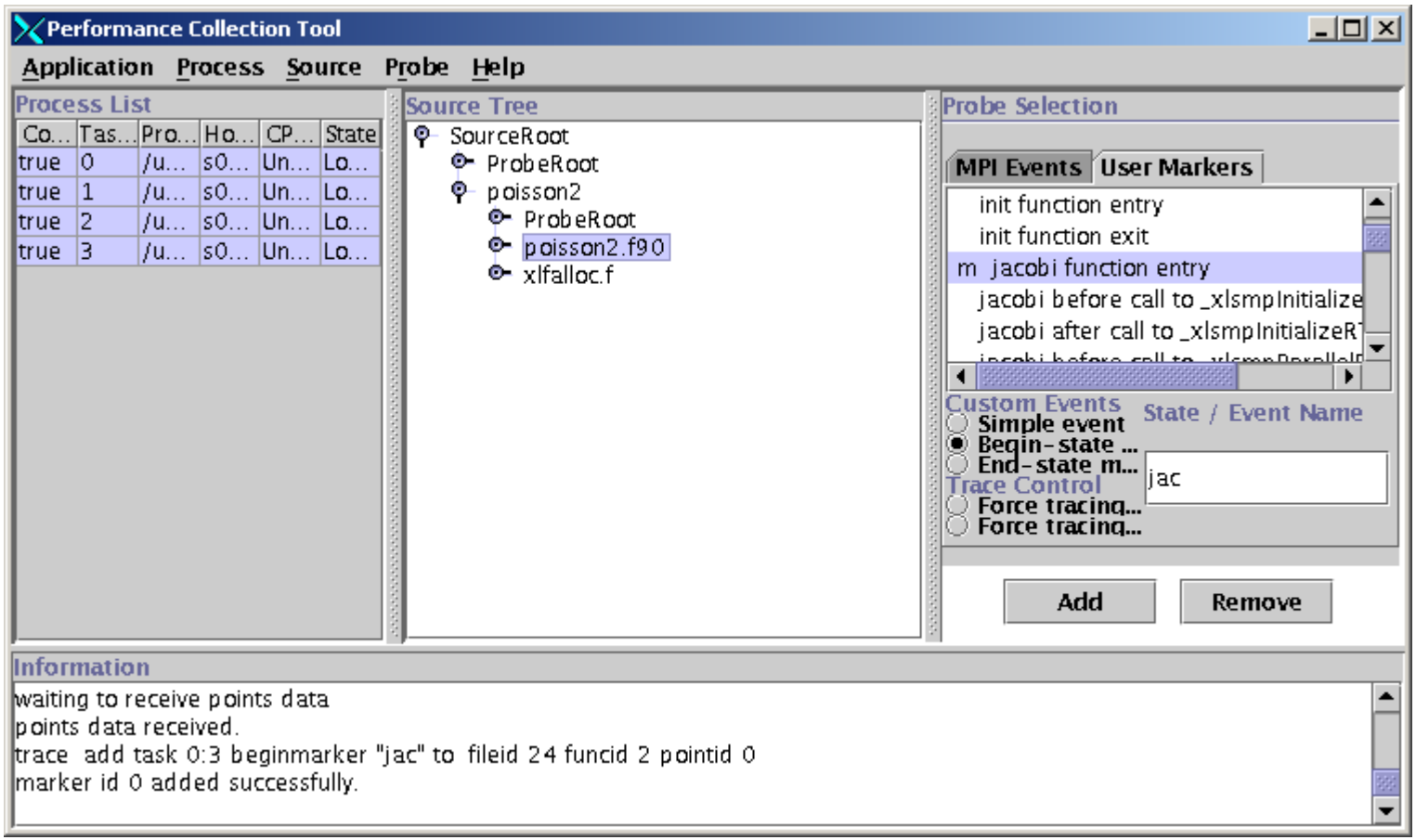
Performance Collection Tool – MPI Events

Selecting MPI Events:

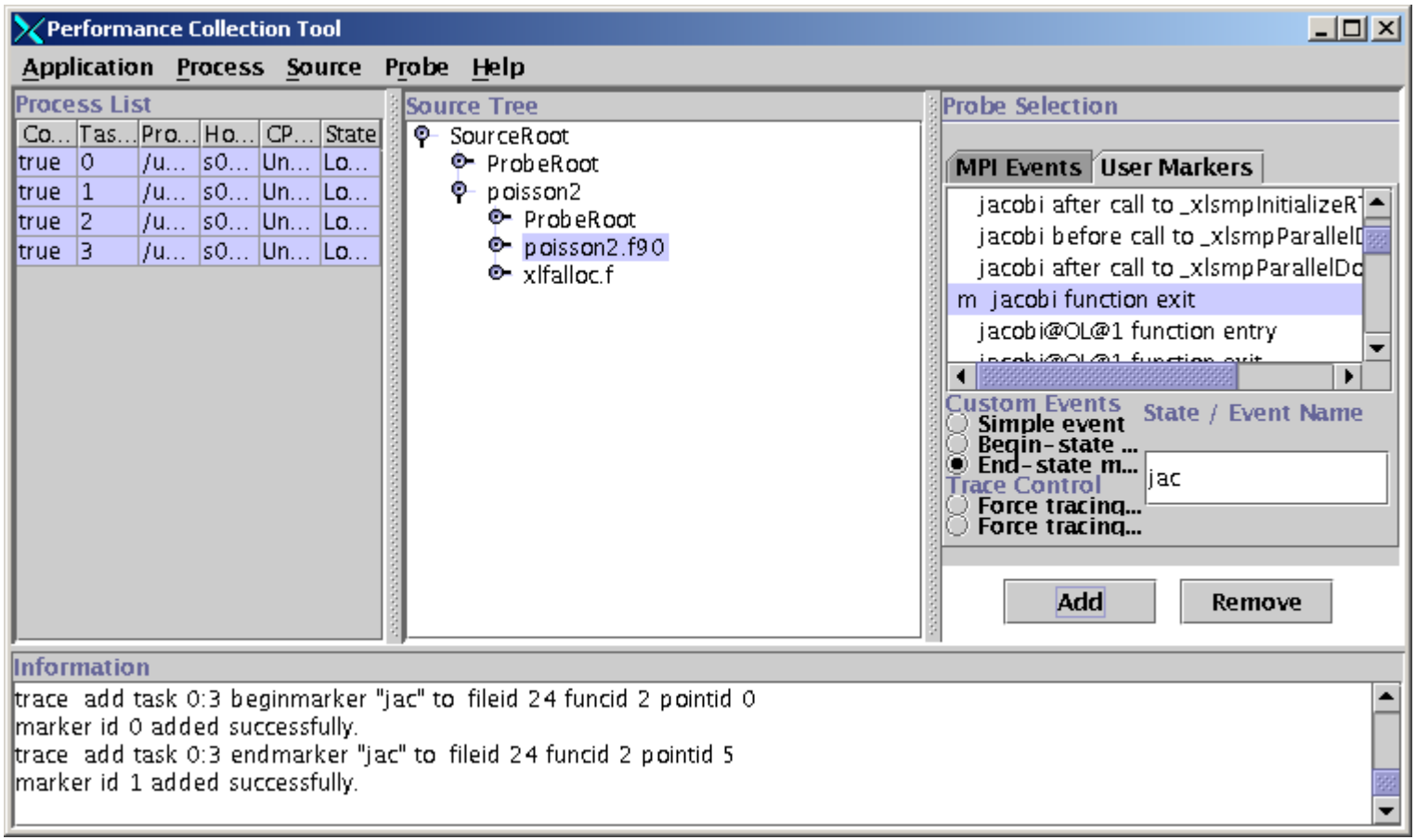


Performance Collection Tool – MPI Events

Selecting User Markers:



Performance Collection Tool – MPI Events



Performance Collection Tool – MPI Events

- After the application completes:
  - Files named *basename.xx* created, one per node (xx is rank of one random task per node)
  - Files are generally large (AIX trace files)
    - Depends on what is traced: mpi, process, idle
  - Need to preprocess files with the *uteconvert* command to produce UTE files
  - Can collect statistics directly from UTE files using *utestats* command, and use the *slogmerge* in order to view with *jumpshot*

Command Line Interface

- Invoke with `pct -c`
- Provide a script file with `pct -c -s scriptfile`

```
load poe exec /usr/common/homes/j/jcarter/poisson/poisson2 poeargs "-procs 4
-nodes 1 -retry 30 -retrycount 30"
select trace
trace set path "/usr/common/homes/j/jcarter/poisson/tmp"
trace set logsize 50
trace set event mpi
trace add mpiname all to file "*"
trace add beginmarker "jacobi" to file "poisson2.f90" funcid 2 pointid 0
trace add endmarker "jacobi" to file "poisson2.f90" funcid 2 pointid 5
trace add beginmarker "update" to file "poisson2.f90" funcid 8 pointid 0
trace add endmarker "update" to file "poisson2.f90" funcid 8 pointid 5
start
wait
```

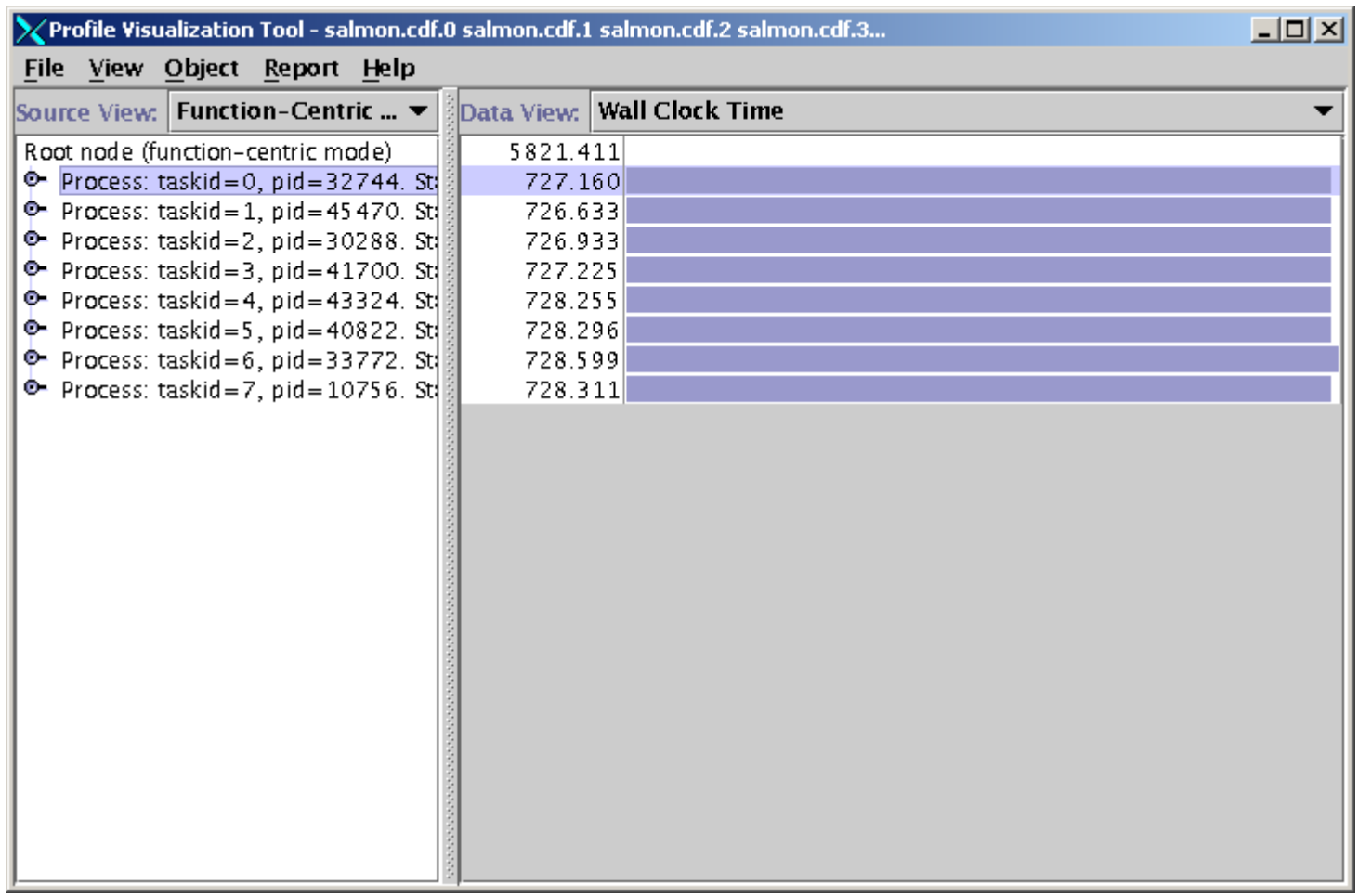
Performance Visualization Tool

- View hardware/OS profiles
- To run

```
module load java
pvt basename.cdf.*
```

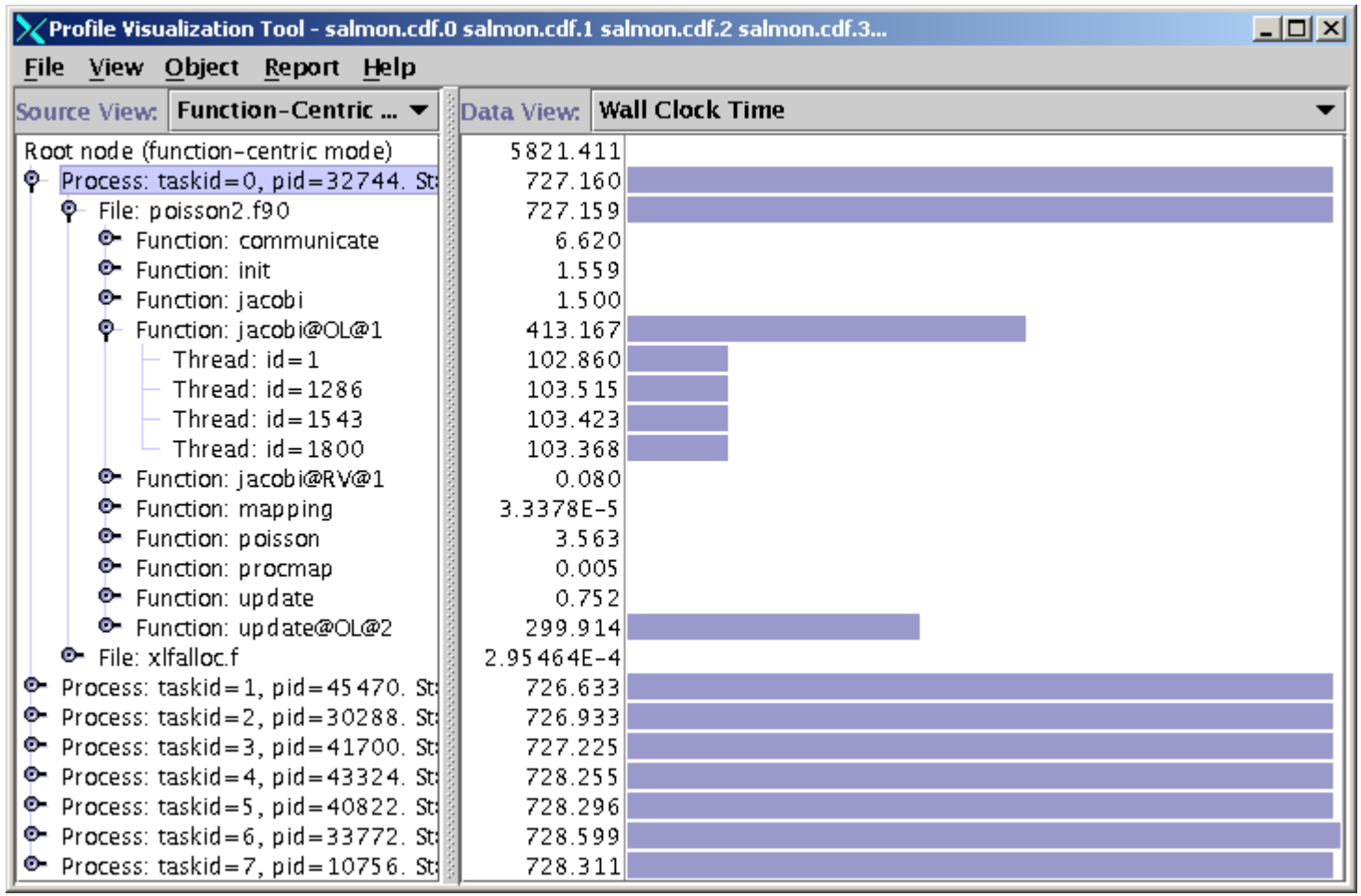
Performance Visualization Tool

Select data to be displayed from "Data View" menu:



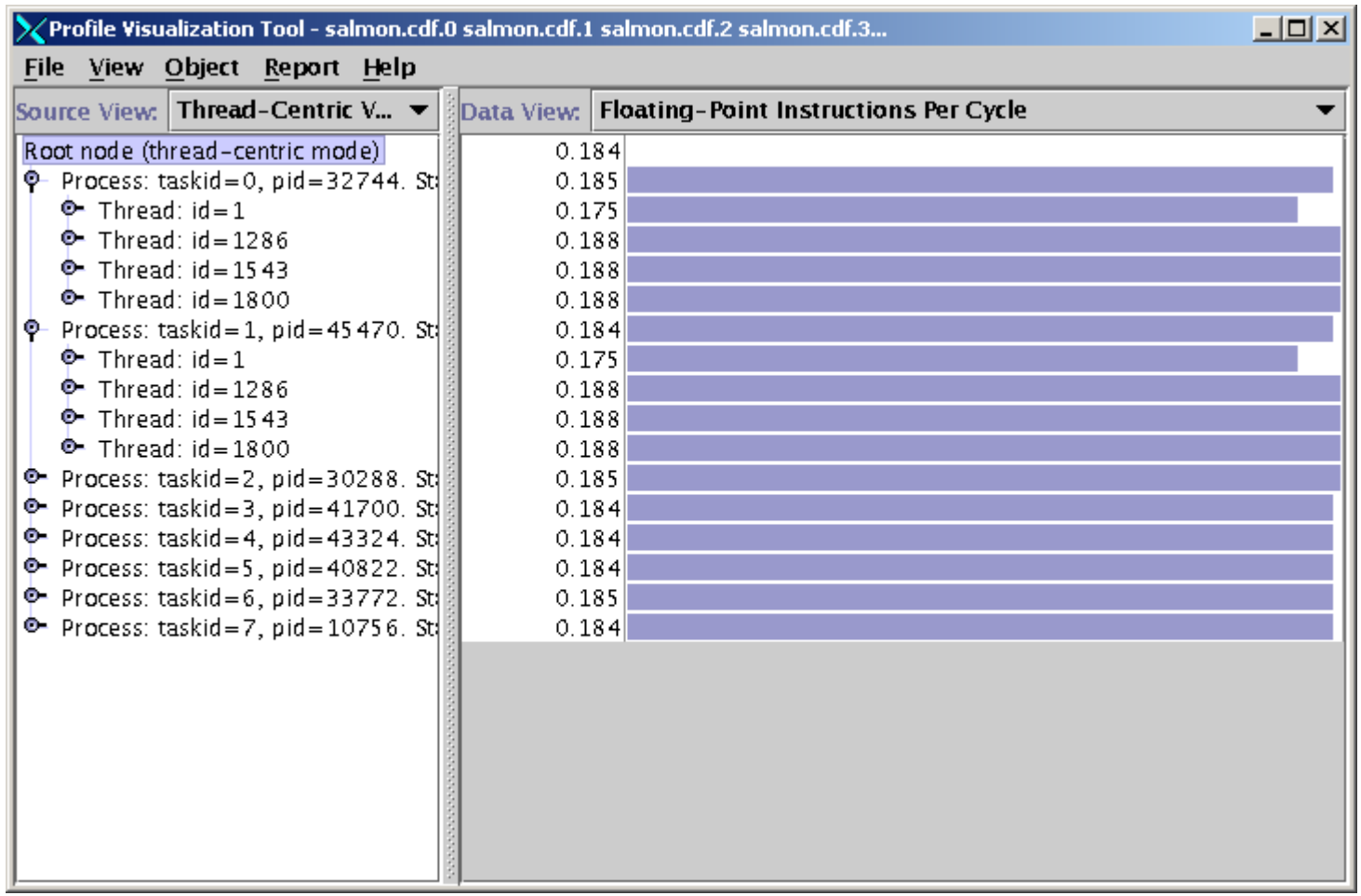
Performance Visualization Tool

Expand processes to show functions and threads:



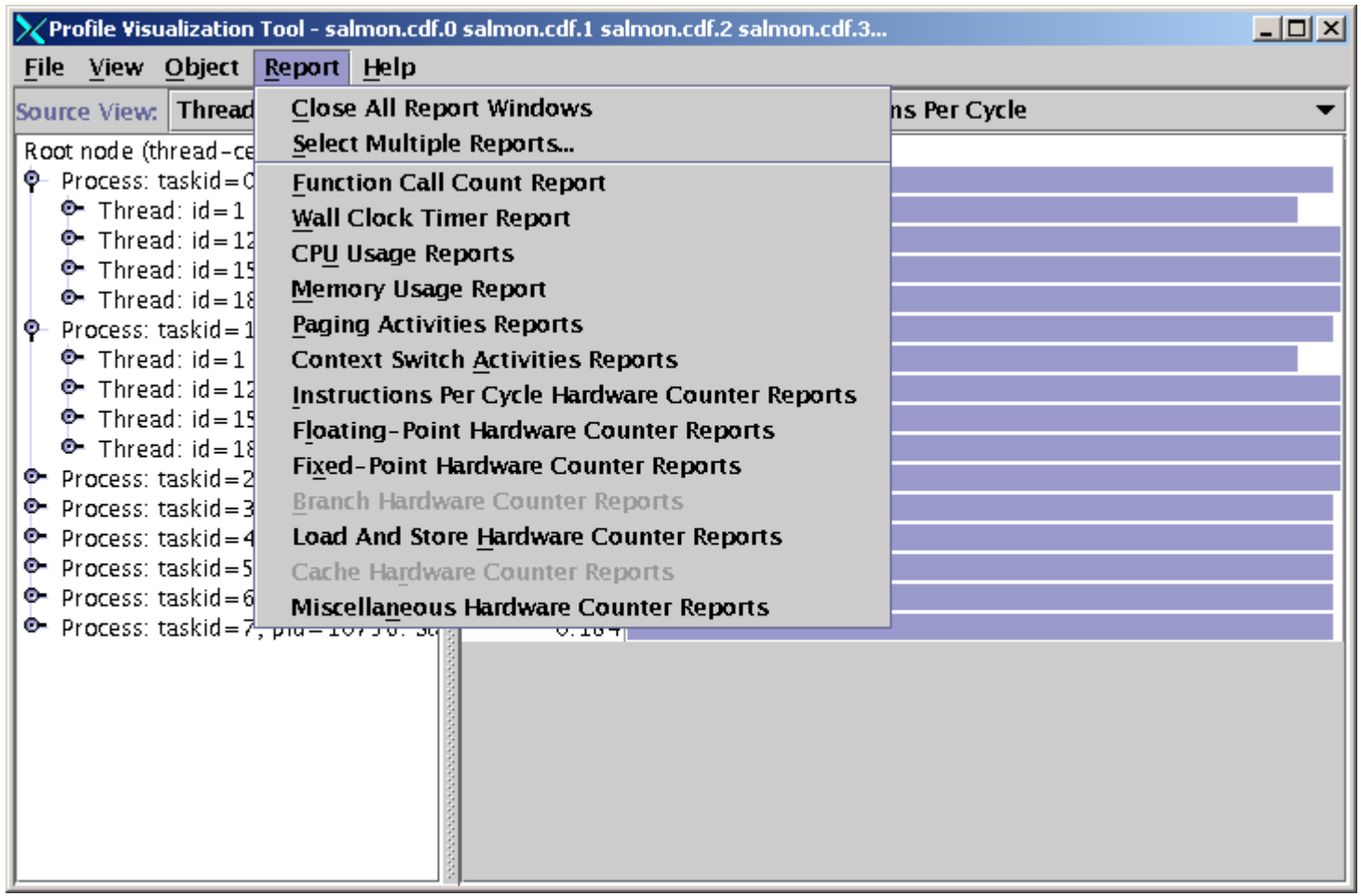
Performance Visualization Tool

Select Thread-Centric View for another way of grouping results:



Performance Visualization Tool

Produce various text reports:



Performance Visualization Tool

Text reports can be saved as tab-separated text files.

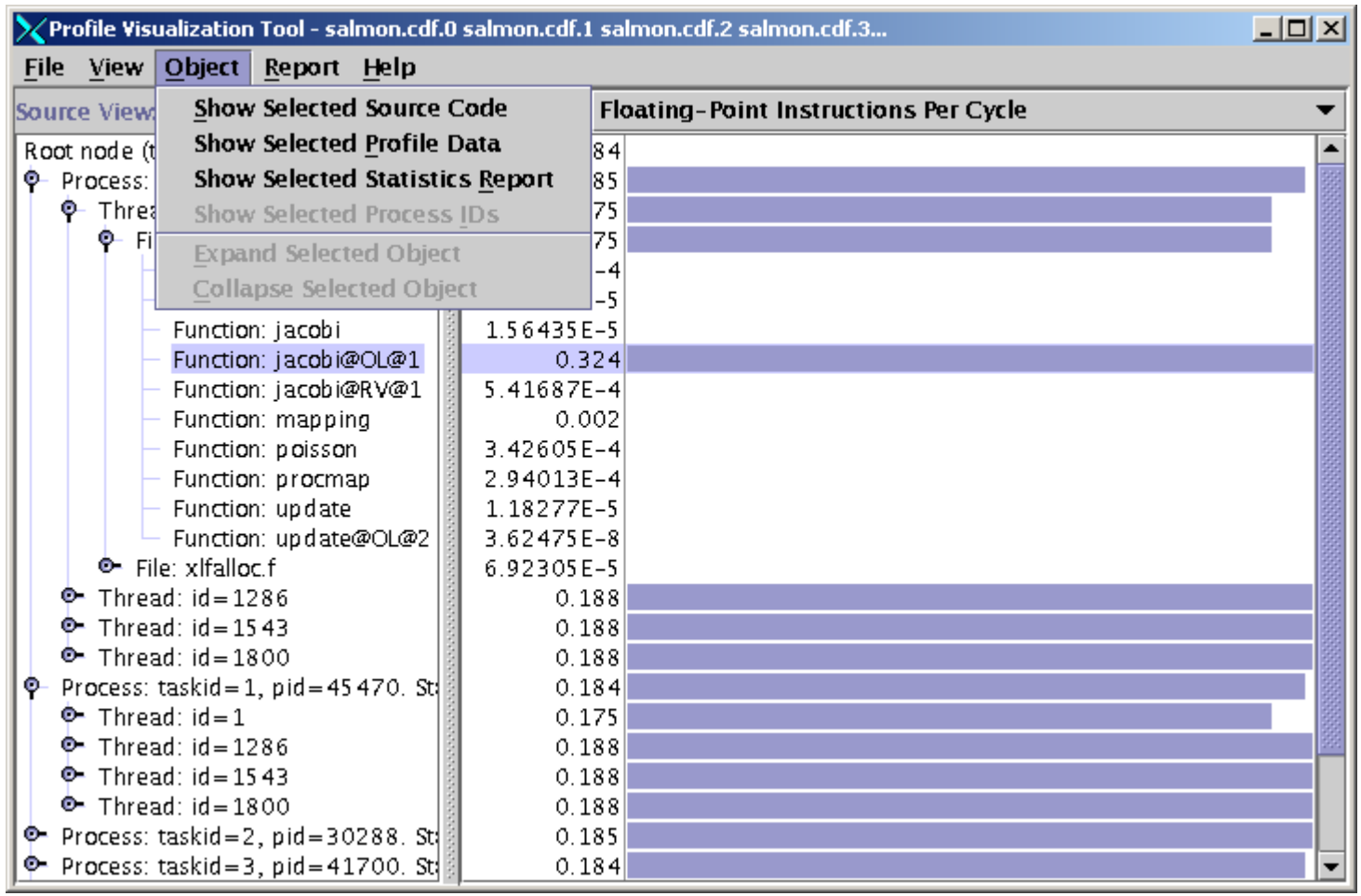
Report: Floating-Point Instructions Per Cycle

File Help

Function Na...	Summary	Average	StdDev	Max	Max ID	Min	Min ID	File Na...
jacobi@OL...	2.594	0.324	2.90580E-4	0.325	2	0.324	7	poisso...
mapping	0.013	0.002	4.31809E-5	0.002	2	0.002	0	poisso...
jacobi@RV...	0.004	5.49161E-4	1.18459E-5	5.76239E-4	1	5.35399E-4	4	poisso...
poisson	0.003	3.27192E-4	8.01559E-6	3.42605E-4	0	3.20347E-4	3	poisso...
procmap	0.002	2.79883E-4	3.05979E-5	3.10977E-4	4	2.25478E-4	2	poisso...
communicate	0.002	2.37299E-4	1.33394E-5	2.54896E-4	4	2.13543E-4	6	poisso...
__xlf61allocg	6.10643E-4	7.63303E-5	3.24355E-6	8.16693E-5	5	6.92305E-5	0	xlfalloc.f
jacobi	1.41628E-4	1.77035E-5	2.63387E-6	2.26101E-5	1	1.43903E-5	4	poisso...
update	7.80023E-5	9.75028E-6	2.22848E-6	1.31864E-5	1	5.68047E-6	6	poisso...
init	6.85003E-5	8.56254E-6	8.55209E-6	1.72078E-5	2	8.54947E-9	1	poisso...
update@O...	4.56823E-7	5.71029E-8	8.99237E-9	7.49644E-8	6	4.89549E-8	1	poisso...

Performance Visualization Tool

Show detailed statistics on one process, function or thread:



Performance Visualization Tool

Function Statistics Report: jacobi@OL@1

File Help

...	this	Summary	Average	StdDev	Max	Max ID	Min	Min ID
Function Call ...	500	16000	2000	0	2000	0	2000	0
Wall Clock Time	102.860	3306.504	413.313	0.578	414.121	4	412.340	2
User CPU Usage	101.660	3263.150	407.894	0.272	408.460	7	407.590	5
System CPU U...	0.930	31.860	3.983	0.247	4.410	6	3.640	2
Maximum Res...	300228	2401464	300183	17.176	300228	0	300172	5
Page Fault Wit...	6114	195592	24449	4.330	24456	5	24442	7
Page Fault Wit...	0	0	0	0	0	0	0	0
Voluntary Con...	0	0	0	0	0	0	0	0
Involuntary Co...	33	26337	3292.125	647.622	4651	4	2488	3
Floating-Point...	0.324	2.594	0.324	2.90580E-4	0.325	2	0.324	7
Fixed-Point In...	0.004	0.035	0.004	2.12258E-4	0.005	5	0.004	2
Branch Instruc...	N/A							
Load Instructi...	0.163	1.306	0.163	1.21356E-4	0.163	2	0.163	7
Store Instructi...	0.041	0.329	0.041	3.06346E-5	0.041	4	0.041	3
Number Of Lo...	6.28826E9	2.01311E11	2.51639E10	8170277.127	2.51748E10	5	2.51546E10	0
FPU1 (Floating...	6.00129E9	1.92042E11	2.40052E10	505732.519	2.40058E10	5	2.40045E10	0
Number Of St...	1.58338E9	5.07191E10	6.33988E9	4307560.447	6.34556E9	5	6.33485E9	0
Processor Clo...	3.85545E10	1.23337E12	1.54171E11	1.38131E8	1.54408E11	7	1.53941E11	2
FPU0 (Floating...	6.49821E9	2.07942E11	2.59928E10	497611.849	2.59935E10	0	2.59922E10	5
FXU2 (Fixed-...	1.88941E7	6.39244E8	7.99056E7	3856520.521	8.49423E7	5	7.54959E7	2
FXU0 (Fixed-...	9.57332E7	2.95926E9	3.69907E8	6.52351E7	4.32752E8	5	2.45477E8	2
FXU1 (Fixed-...	4.59100E7	1.82913E9	2.28641E8	5.93726E7	3.42402E8	4	1.83807E8	0

Visualizing MPI Events

- Convert AIX tracefiles to UTE tracefiles

  `uteconvert -n` *`n basename`*`.`

- Merge UTE tracefiles

  `utemerge -n` *`n basename`*`.ute.`

- Generate statistics file

  `utestats -o` *`basename`*`.stats` *`basename`*`.ute.ute`

- Generate slog file for `jumpshot` tool

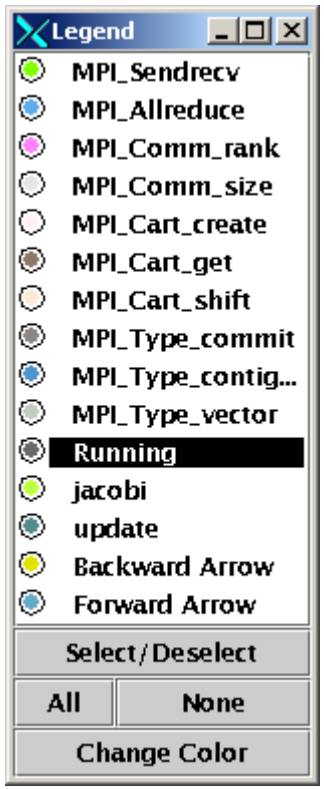
  `slogmerge -n` *`n basename`*`.ute.`

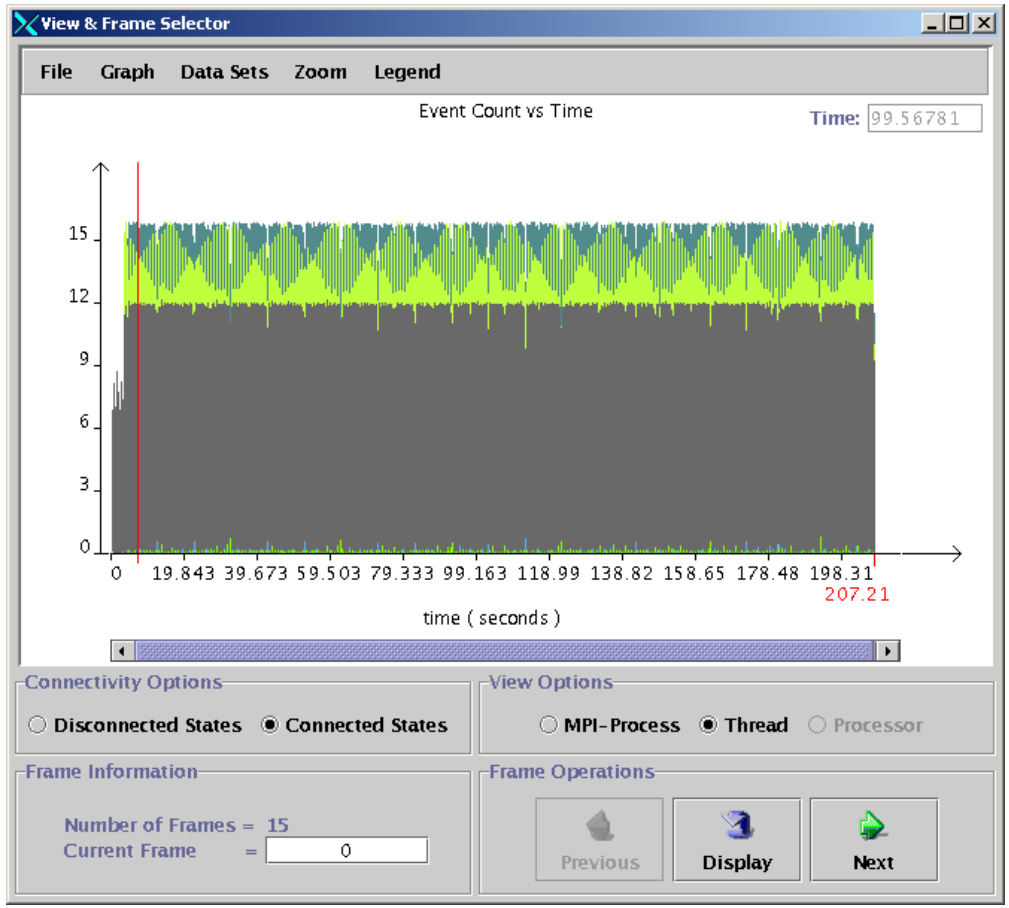
- Run jumpshot (limit is 64 MPI processes)

  ```
  module load java mpe
  jumpshot basename.slog
  ```

jumpshot

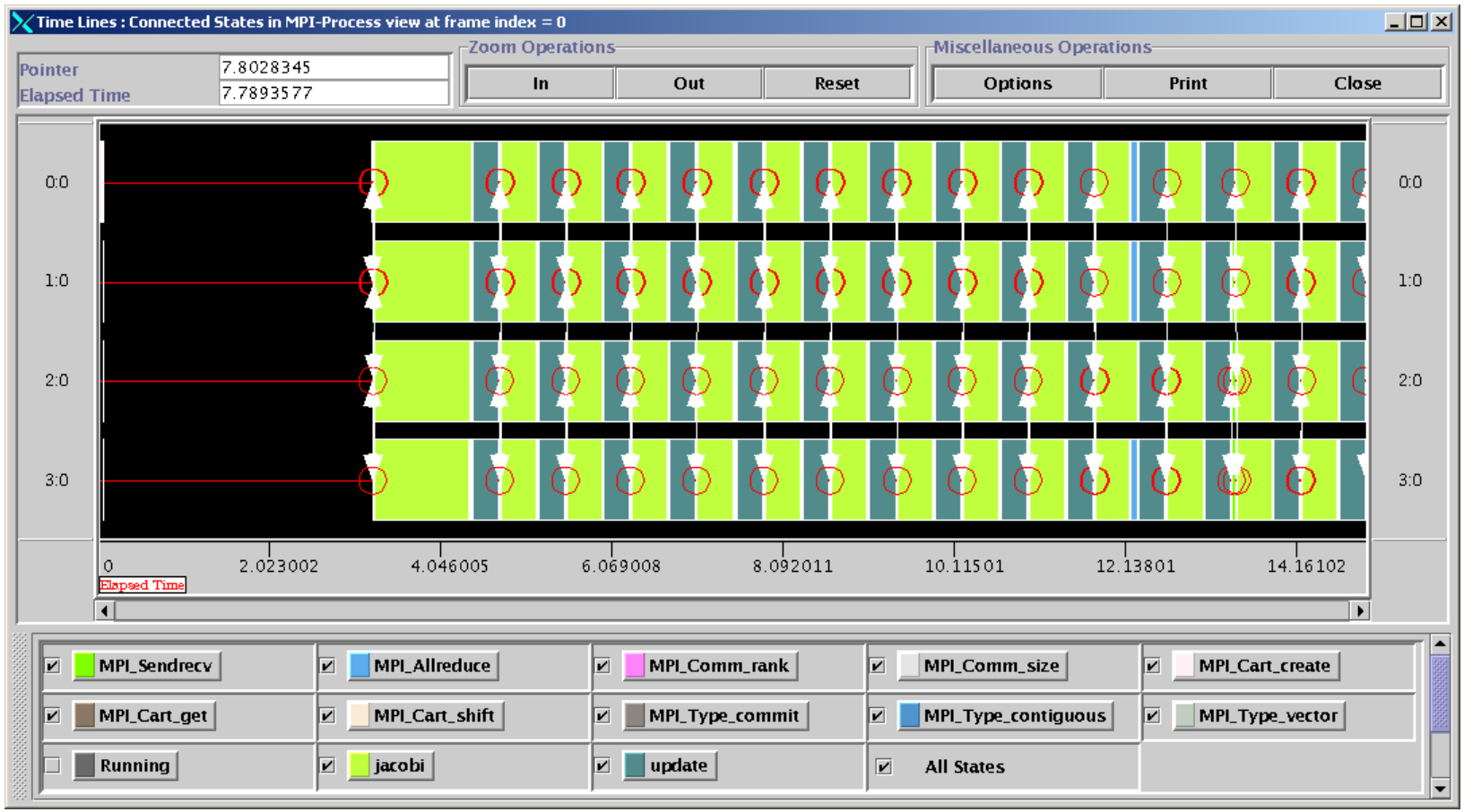
Entire profile is divided into frames, select frame, select MPI-Process or Thread, then click "Display".



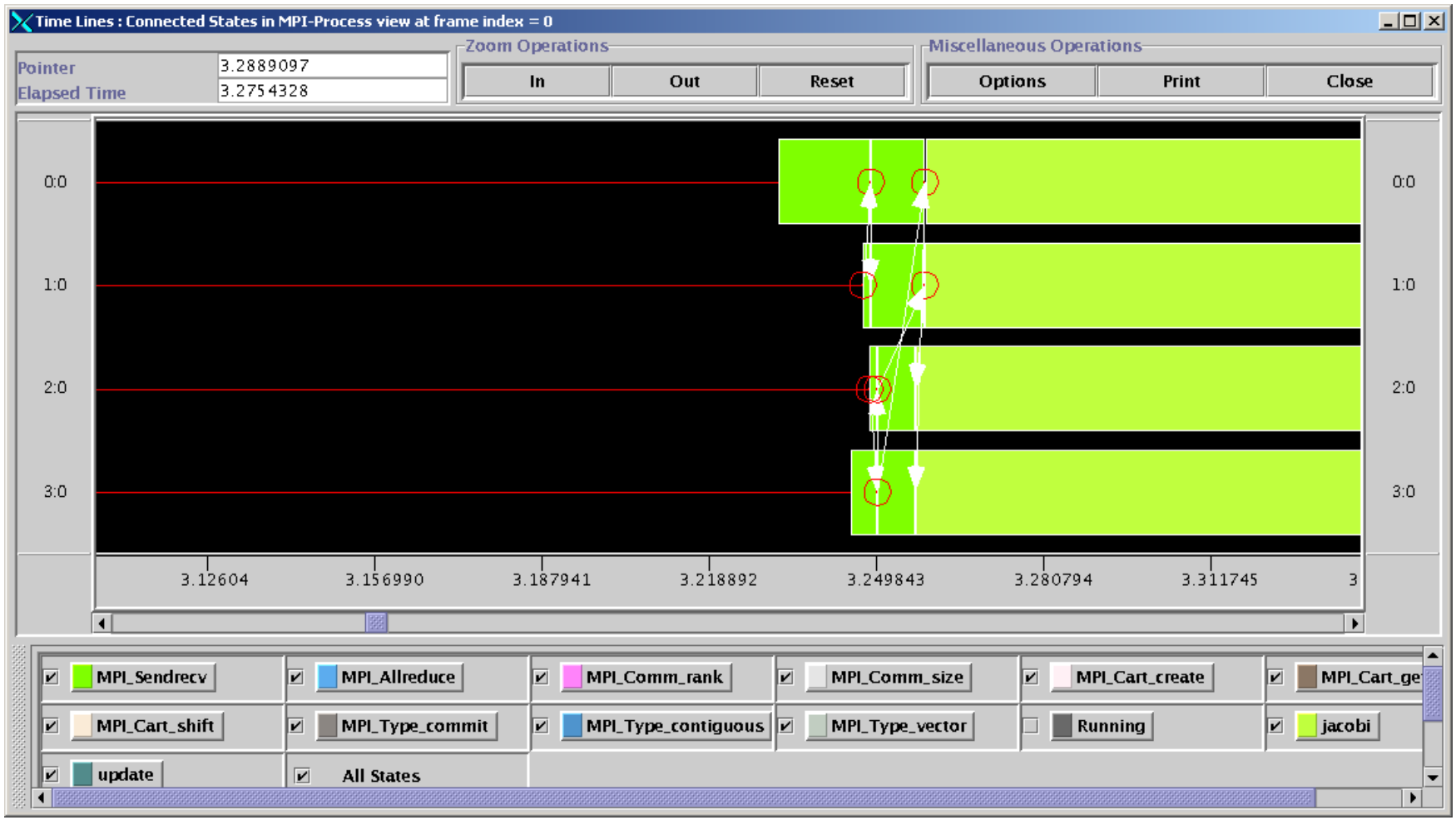


jumpshot

Basic MPI process view, use zoom buttons to see detail

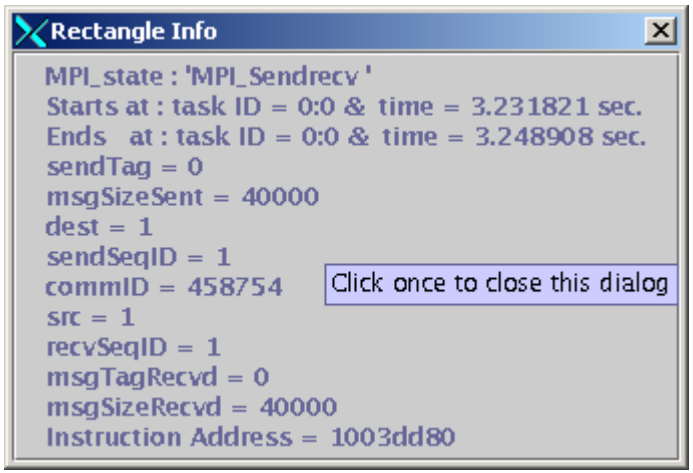


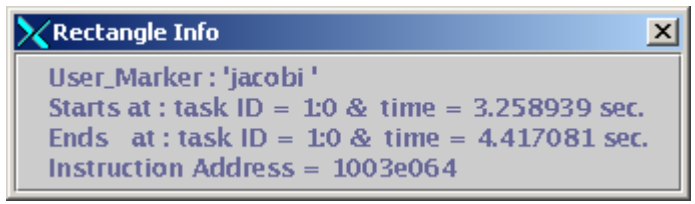
jumpshot


Time Lines : Connected States in MPI-Process view at frame index = 0
Pointer
3.2889097
Elapsed Time
3.2754328
Zoom Operations
In
Out
Reset
Miscellaneous Operations
Options
Print
Close
0:0
1:0
2:0
3:0
3.12604
3.156990
3.187941
3.218892
3.249843
3.280794
3.311745
MPI_Sendrecv
MPI_Allreduce
MPI_Comm_rank
MPI_Comm_size
MPI_Cart_create
MPI_Cart_shift
MPI_Type_commit
MPI_Type_contiguous
MPI_Type_vector
Running
jacobi
update
All States

jumpshot

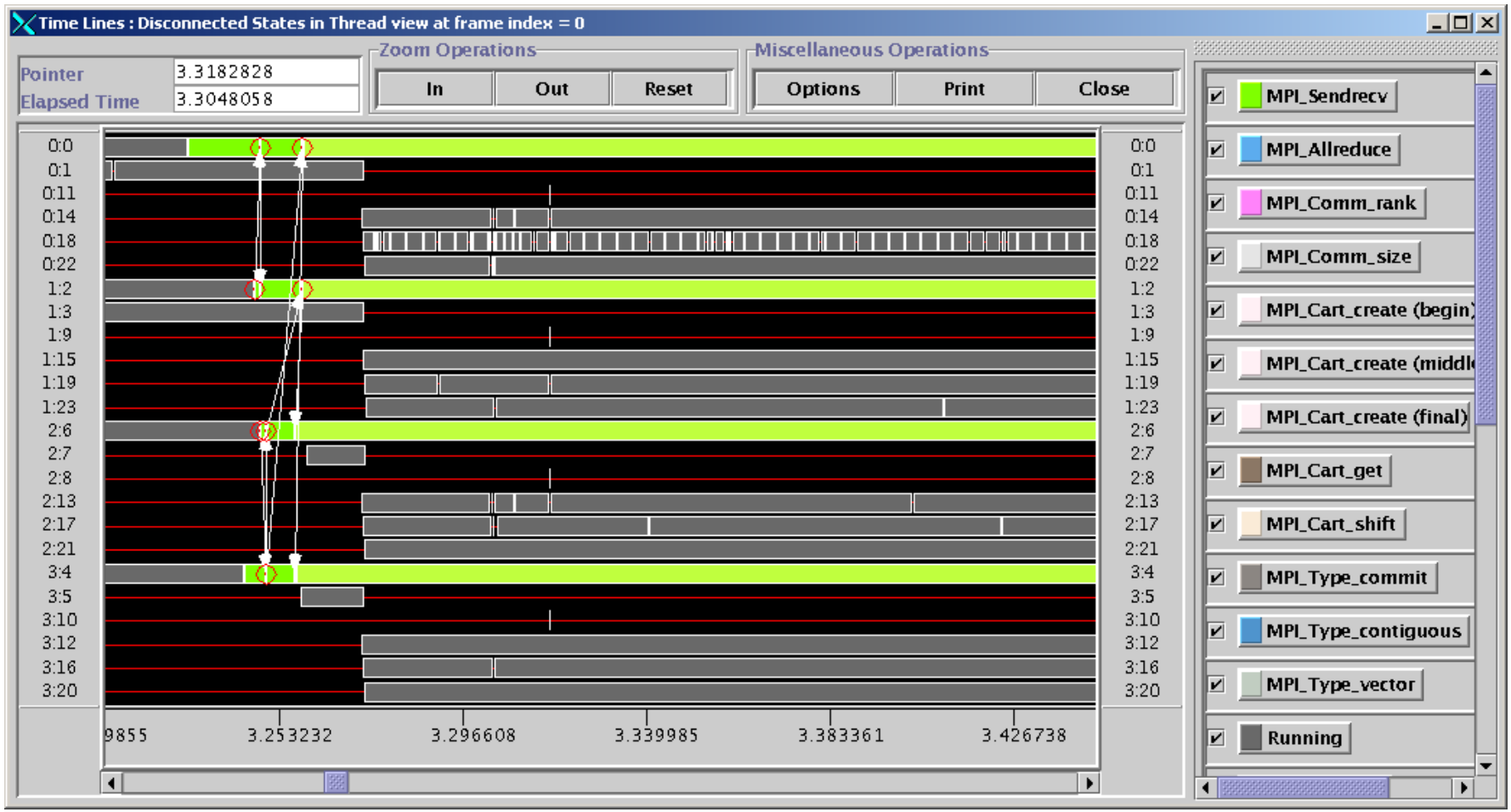
Clicking on messages (red circles) or blocks produces a detailed text description:





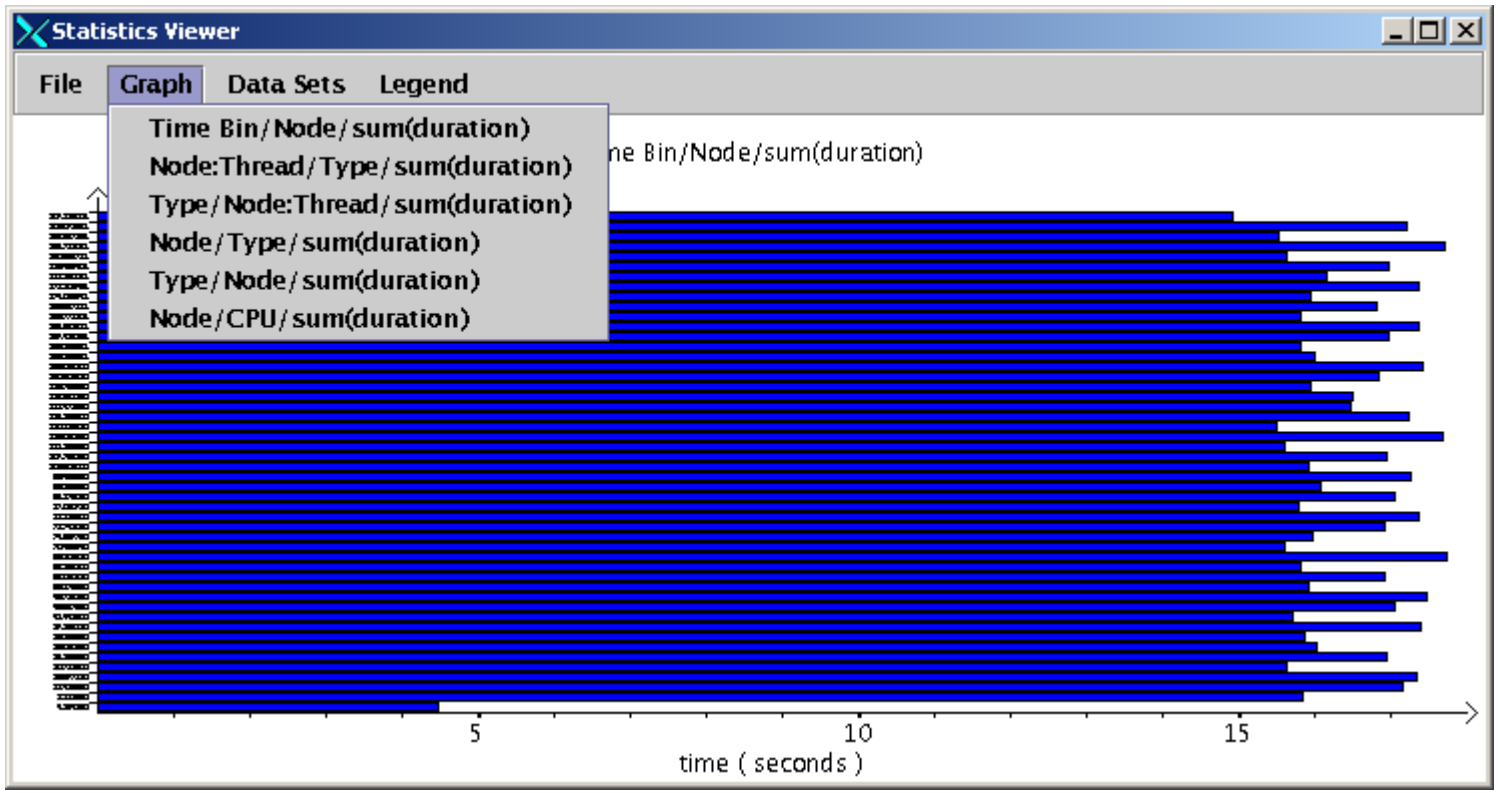
jumpshot

Thread level view:

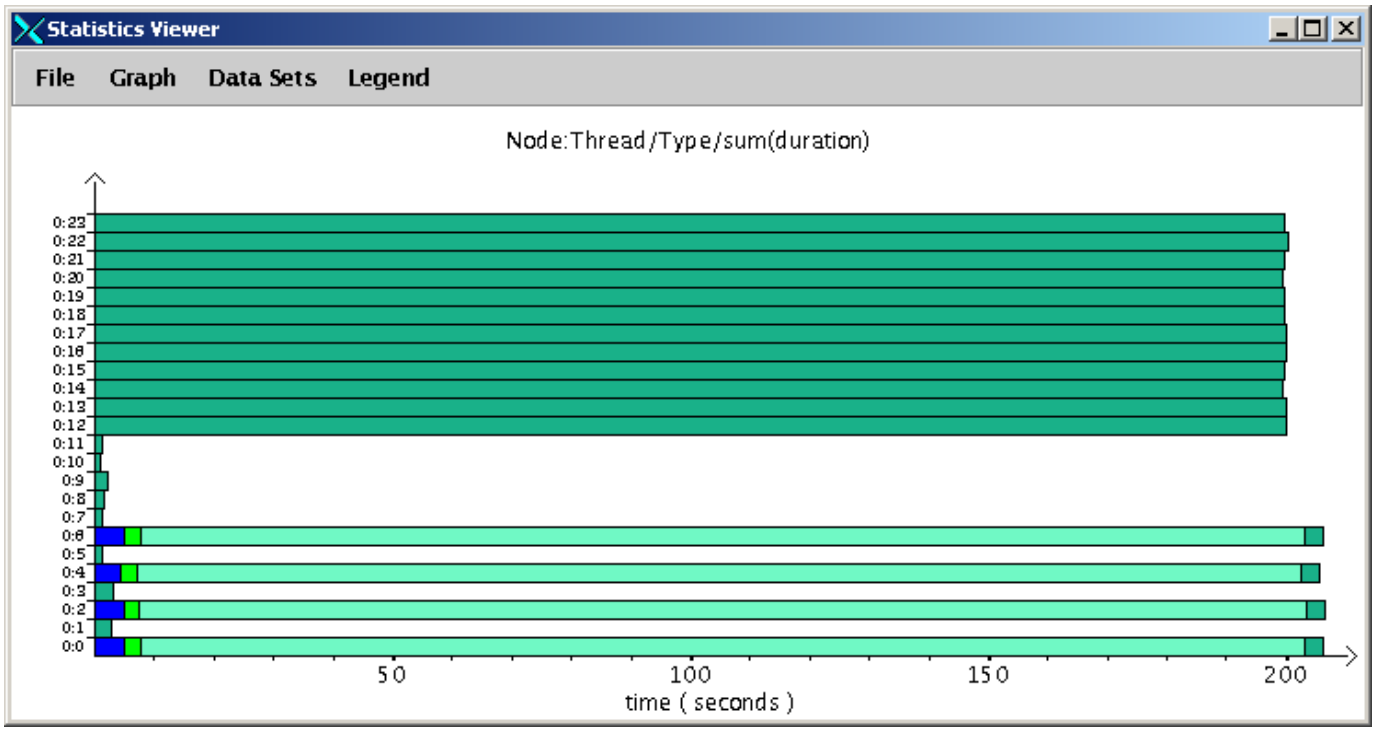


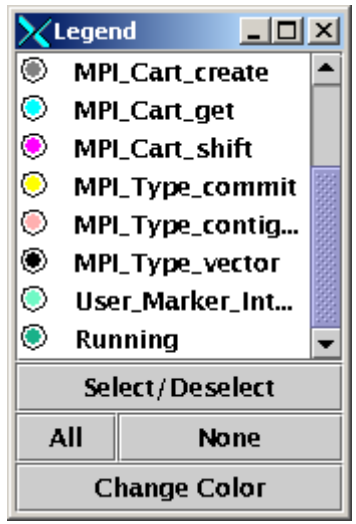
jumpshot

To visualize statistics file. Go back to "View and Frame Selector Window", use "File->View Statistics" menu to open the "Statistics Viewer" window, and from there open previously generated "stats" file.



jumpshot


Statistics Viewer
File
Graph
Data Sets
Legend
Node:Thread/Type/sum(duration)
0:23
0:22
0:21
0:20
0:19
0:18
0:17
0:16
0:15
0:14
0:13
0:12
0:11
0:10
0:9
0:8
0:7
0:6
0:5
0:4
0:3
0:2
0:1
0:0
50
100
150
200
time (seconds)


Legend
MPI_Cart_create
MPI_Cart_get
MPI_Cart_shift
MPI_Type_commit
MPI_Type_contig...
MPI_Type_vector
User_Marker_Int...
Running
Select/Deselect
All
None
Change Color

Further Reading

- IBM PE for AIX, Operation and Use, Volume 2
  - Chapter 3 and Appendix A
  - http://hpcf.nersc.gov/vendor_docs/ibm/pe/am103mst.html
- Tour of Jumpshot
  - /usr/common/usg/mpe/1.2.2/share/jumpshot-3/doc/TourStepByStep.pdf

CEPBA Tools

- Run application under the control of *ompitrace*
  - Uses DPCL technology to insert probes into executable
- Merge trace files with *omp2prv*
- Visualize results with *paraver*

ompitrace

- ompitrace [-v] [-locks] [-nosw] [-r] [functions:[all|*funcfile*]]
  [-calls:*callfile*] [-stdin:*stdin*] [-stdout:*stdout*]
  [-stderr:*stderr*] [-counters] *FullPathPOE*
  FullPathApplName [*ApplPOEParams*]
- Options:
  - -v Verbose mode
  - -locks Trace lock calls
  - -nosw Not use the switch clock.
  - -r Must be set if application uses threaded MPI library
  - -function Extra functions to be traced (user code)
  - -calls Extra calls to be traced (system code)

ompitrace

  - -counters:[*option*,...] Trace counters automatically, options:
    - **user**/nouser
    - **parallel**/noparallel
    - **calls**/nocalls
    - mpi/**nompi**

- ompitrace -list -functions:*funcfile* -calls:*callfile* *FullPathPOE FullPathApplName* [*ApplPOEParams*]
  - -list List all traceable application functions/calls.
- ompitrace -hwc [*counter*]
  - -hwc [*counter*] List the event counters available for each counter. If a counter is selected, event are listed only for the selected ones.

Related environment variables

- `MPTRACE_ADDFUNCTIONS`: Filename containing additional functions to trace.
- `MPTRACE_DIR`: Output temporary/tracefile directory.
- `MPTRACE_BUFFER_SIZE`: Number of events allowed in thread event buffers.
- `MPTRACE_FILE_SIZE`: Maximum trace file size in Mbytes
- `MPTRACE_COUNTERS`: Hardware event traced for each hardware counter. Can use `$MPTRACE_FPHWC` to use same set as *hpmcount* command.
- `MPTRACE_LABELS`: User manually event traced labels.

ompitrace

- ompitrace is a powerful tool to capture the states and events of a program:
  - Each thread passes through many states as it executes. The two basic states are *running* and *idle*. Also recognized are states that are associated with OpenMP (thread) or MPI actions, e.g. scheduling a thread, waiting for a message, collective operation.
  - Data can be recorded at certain execution events. Typical events are the entry and exit of a threaded-routine, the entry and exit of an MPI routine. You can add the entry and exit of any user-written routines. When an event occurs, data related to event are saved along with a time stamp.

ompitrace

```
$ module load cepba
$ export MPTRACE_COUNTERS=${MPTRACE_FPHWC}
$ ompitrace -r -nosw -counters:parallel poe poisson2 -procs 4 -nodes 1
OMPItrace tool (Version 1.1)

Tracing application: poisson2

Tracing Parameters:
  * Local clock
  * Tracing counters: user,parallel,calls,nompi

Parameters:
  0: /usr/bin/poe
  1: /usr/common/homes/j/jcarter/poisson/poisson2
  2: -procs
  3: 4
  4: -nodes
  5: 1
```

ompitrace

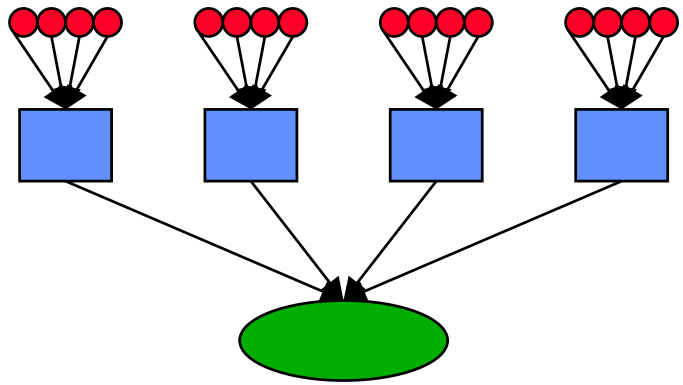
```
-> Creating the target application <-
subfilter: default repo mpccc will be charged
llsubmit: Processed command file through Submit Filter:
    "/usr/common/nsg/etc/subfilter".
-> Expanding source files <-
        -> ( 0 ) Expanding: poisson2.f90
        -> ( 1 ) Expanding: xlfalloc.f
-> Loading module (32-bit): /usr/common/usg/cepba2/etc/openmp+mpi32_nsw <-
Loading module in MPI application ...
Module loaded.
-> Searching probes <-
-> Installing probes ( 33 ) <-
-> Activating probes <-
EndProbeCount: 2
-> Starting application <-
 . . .
ompitrace: application has been traced
ompitrace: remember to merge intermediate traces by using ompi2prv tool:
              ompi2prv *.mpit -s *.sym -o poisson242434.prv
```

ompi2prv

```
$ ompi2prv *.mpit -s *.sym -o poisson245306.prv
Getting input arguments :
        * file (0) poisson2000004530600000000000.mpit will be APPL 1 TASK 1 THREAD 1
    (Pid 45306)
        * file (1) poisson2000004530600000100000.mpit will be APPL 1 TASK 2 THREAD 1
    (Pid 45306)
        * file (2) poisson2000004530600000200000.mpit will be APPL 1 TASK 3 THREAD 1
    (Pid 45306)
        * file (3) poisson2000004530600000300000.mpit will be APPL 1 TASK 4 THREAD 1
    (Pid 45306)
SymFile = poisson20000045306.sym (ApplNum = 0)

ompi2prv tool (Version 1.1)

Generating paraver tracefile poisson245306.prv... (Number of applications = 1 Number
    of traces = 4)

Search_CommunicatorDefinitions ...
Rewind traces ...
. . . .
Paraver trace file poisson245306.prv has been generated
```

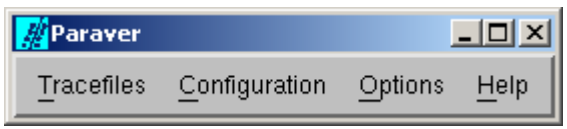
paraver

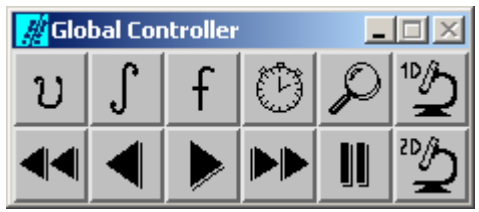
- Paraver is a powerful tool to visualize states and events in tracefiles
- Each user thread has a trace, the values of states (or events) in several thread traces can be merged into a trace for an MPI task
- MPI tasks can be merged into an application

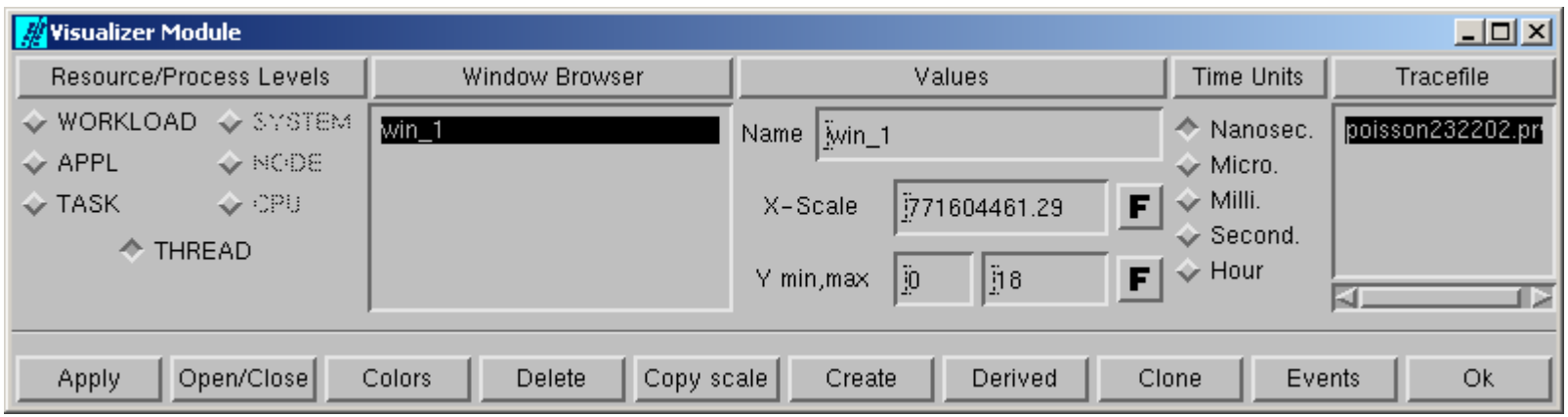


paraver

Three windows (and the display window) are created.

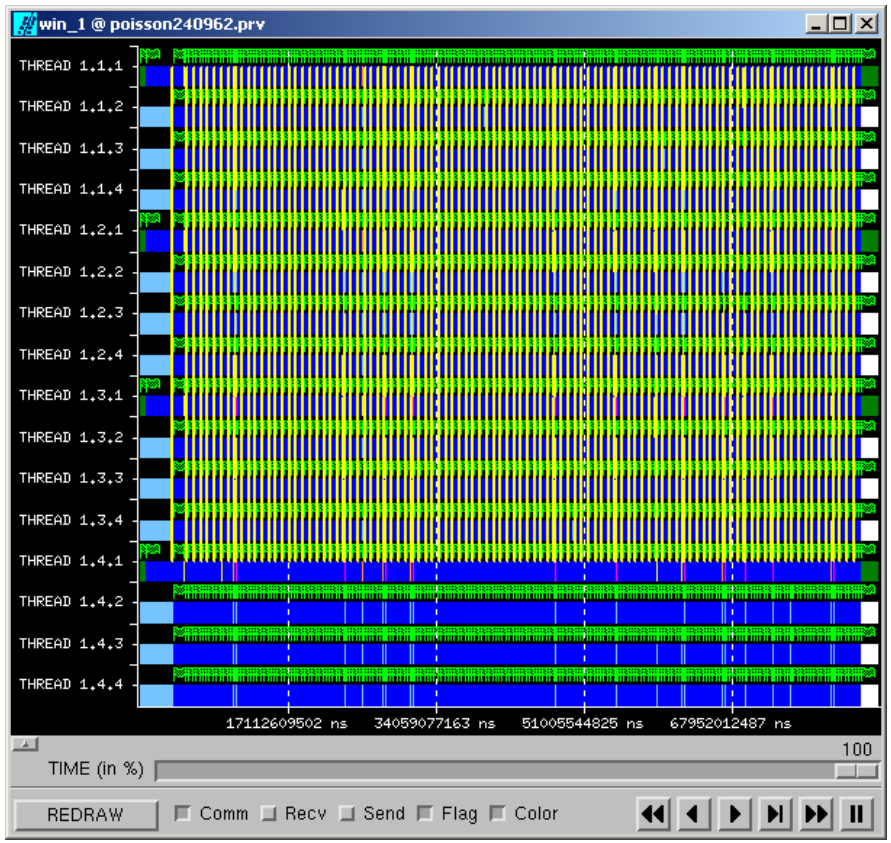






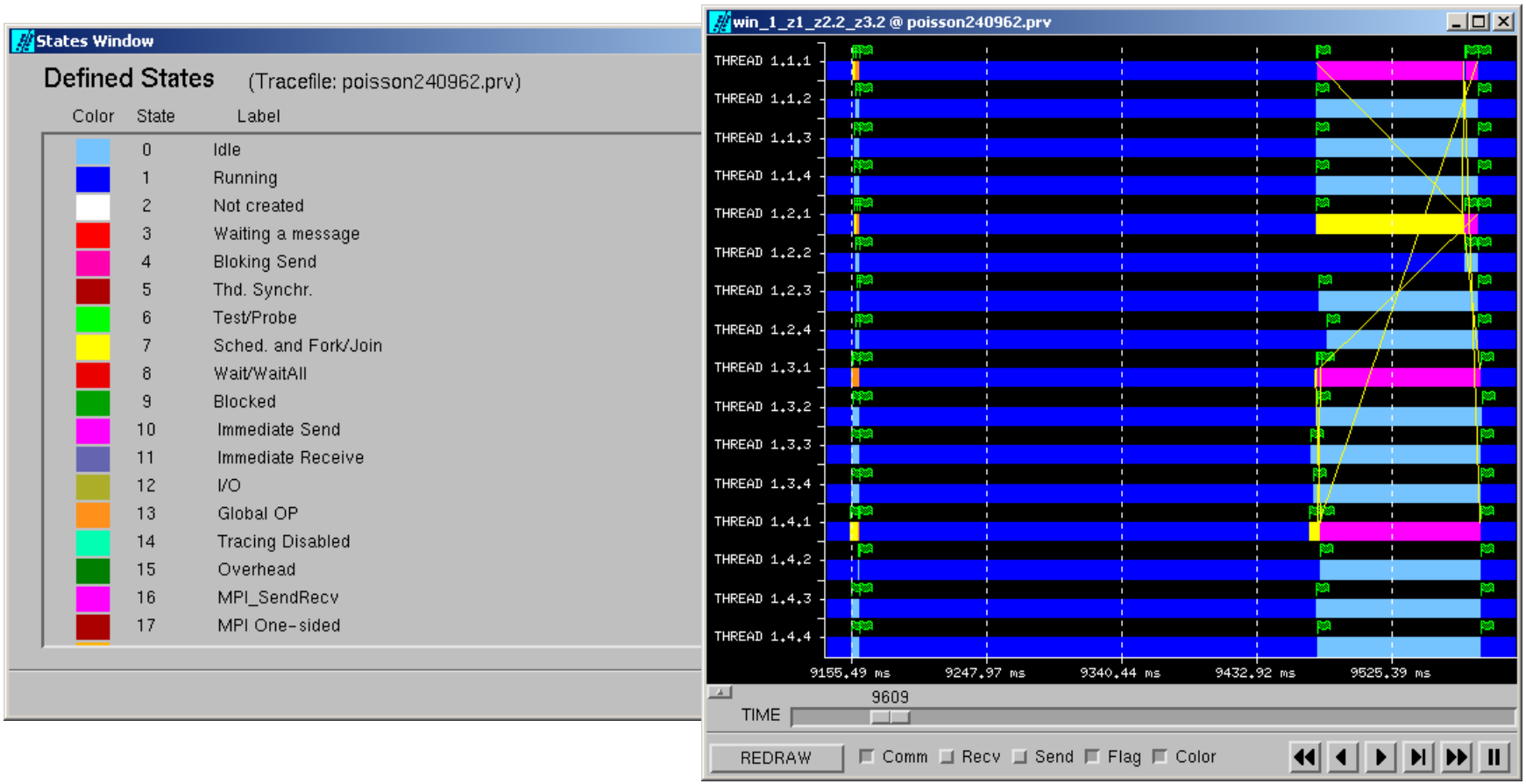
paraver

The default display window is usually too cluttered to make out any detail.
Zoom in by right clicking and choose "Zoom" from the menu.
Or click the magnifying glass in the Global Controller window.
Two additional clicks define the area to be displayed.

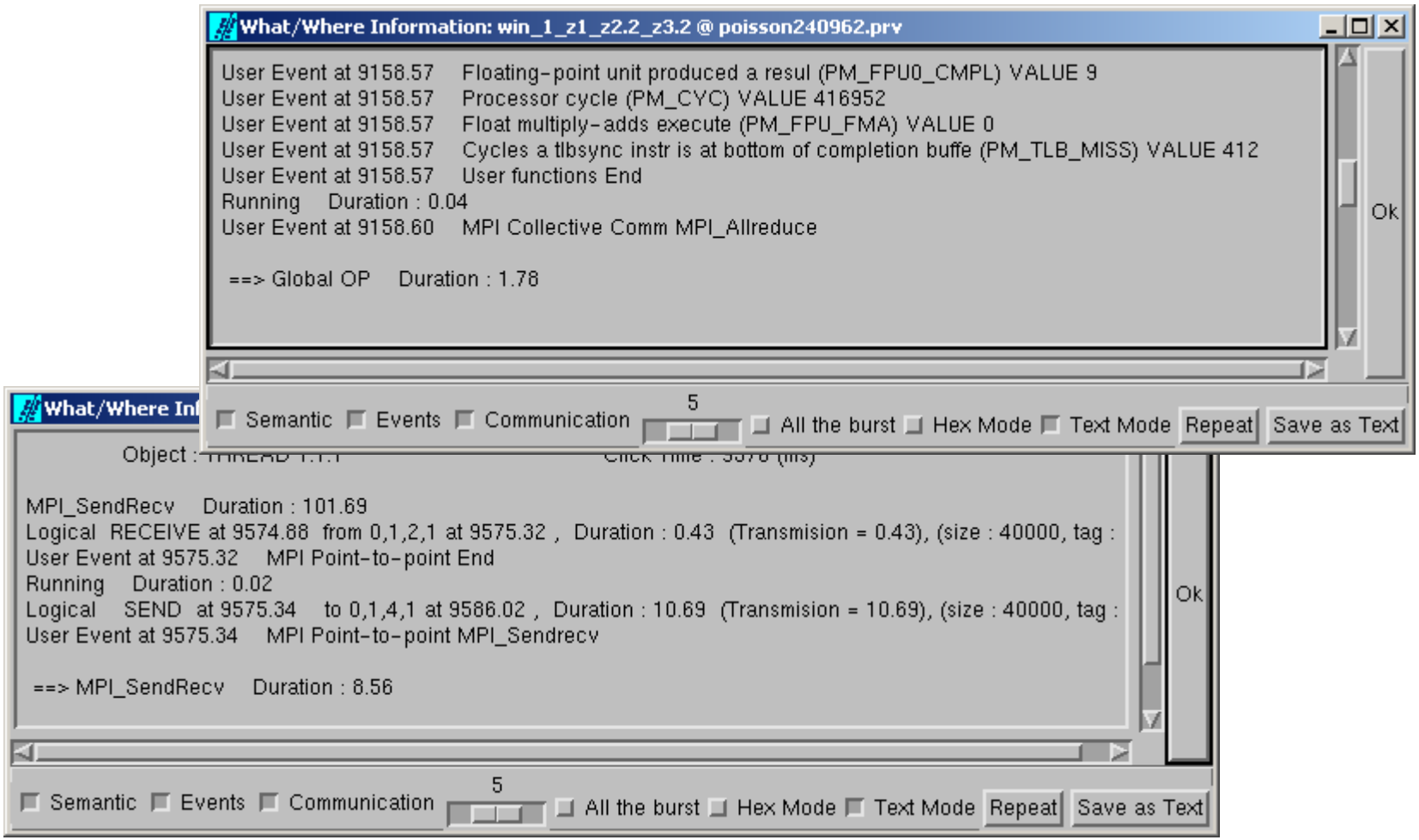


paraver

From "paraver" window, Use "Tracefiles->Trace Information->Defined States window" to bring up a window to show the state to color mapping.

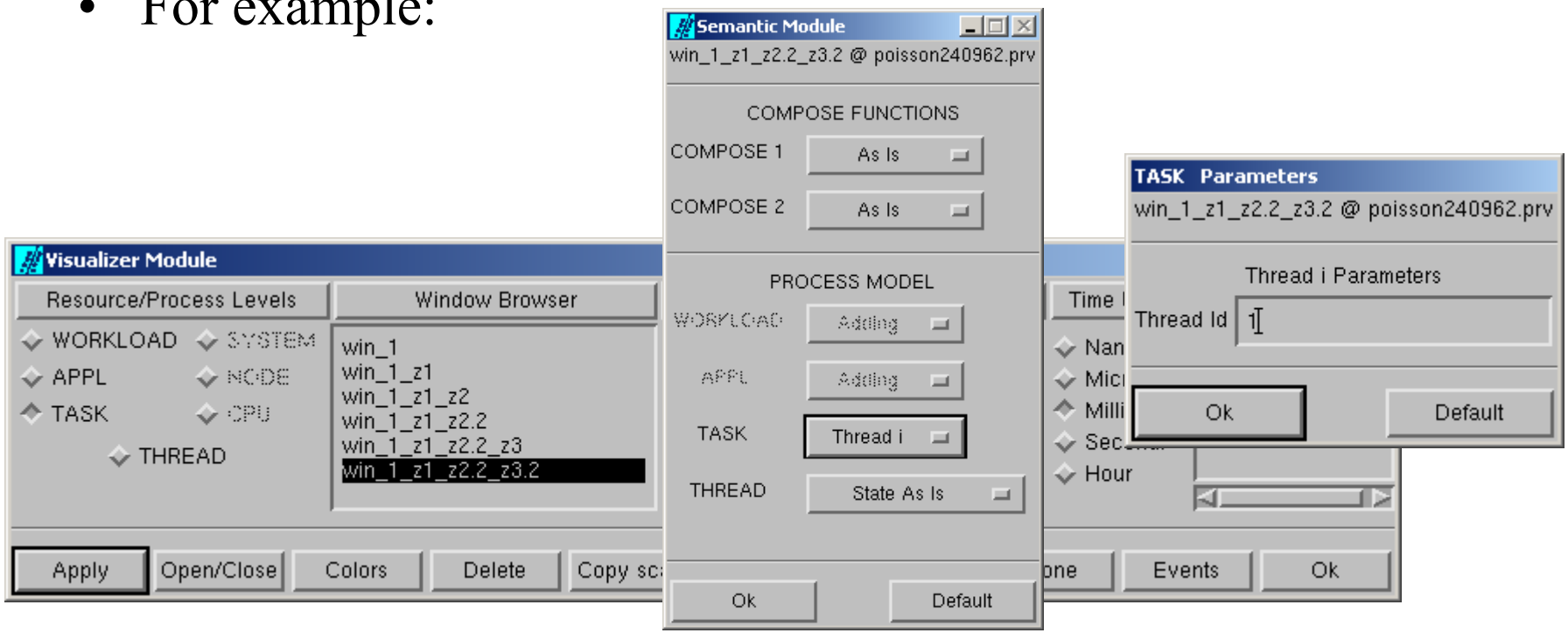


paraver


What/Where Information: win_1_z1_z2.2_z3.2 @ poisson240962.prv
User Event at 9158.57 Floating-point unit produced a resul (PM_FPU0_CMPL) VALUE 9
User Event at 9158.57 Processor cycle (PM_CYC) VALUE 416952
User Event at 9158.57 Float multiply-adds execute (PM_FPU_FMA) VALUE 0
User Event at 9158.57 Cycles a tlbsync instr is at bottom of completion buffe (PM_TLB_MISS) VALUE 412
User Event at 9158.57 User functions End
Running Duration : 0.04
User Event at 9158.60 MPI Collective Comm MPI_Allreduce
==> Global OP Duration : 1.78
Ok
Semantic Events Communication 5 All the burst Hex Mode Text Mode Repeat Save as Text
What/Where In
MPI_SendRecv Duration : 101.69
Logical RECEIVE at 9574.88 from 0,1,2,1 at 9575.32 , Duration : 0.43 (Transmision = 0.43), (size : 40000, tag :
User Event at 9575.32 MPI Point-to-point End
Running Duration : 0.02
Logical SEND at 9575.34 to 0,1,4,1 at 9586.02 , Duration : 10.69 (Transmision = 10.69), (size : 40000, tag :
User Event at 9575.34 MPI Point-to-point MPI_Sendrecv
==> MPI_SendRecv Duration : 8.56
Ok
Semantic Events Communication 5 All the burst Hex Mode Text Mode Repeat Save as Text

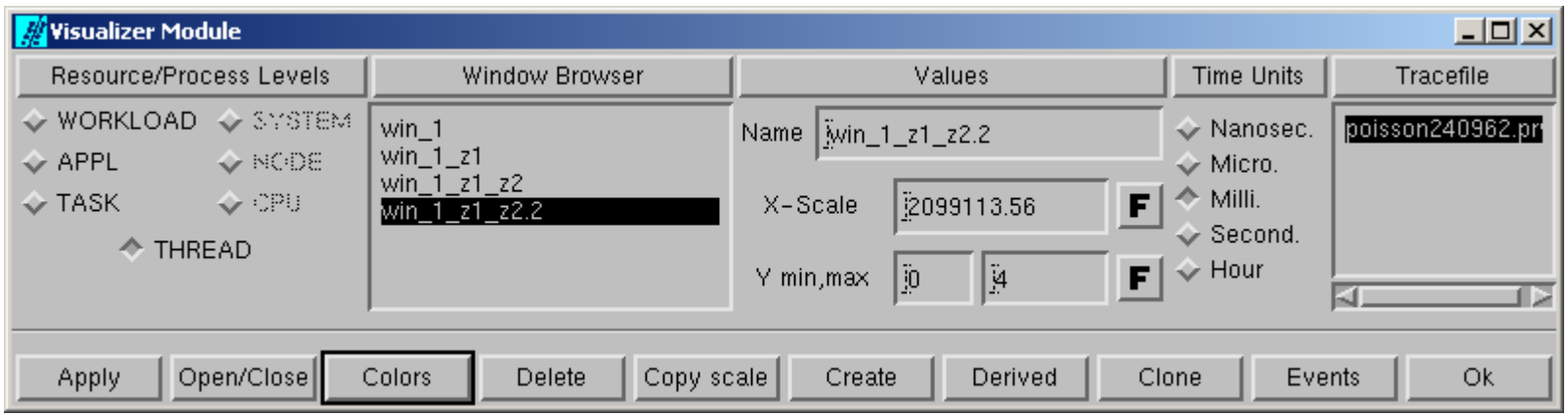
paraver

- Paraver enables you to construct customized displays, but you have to become familiar with the Visualizer, Semantic and Filter modules.
- For example:

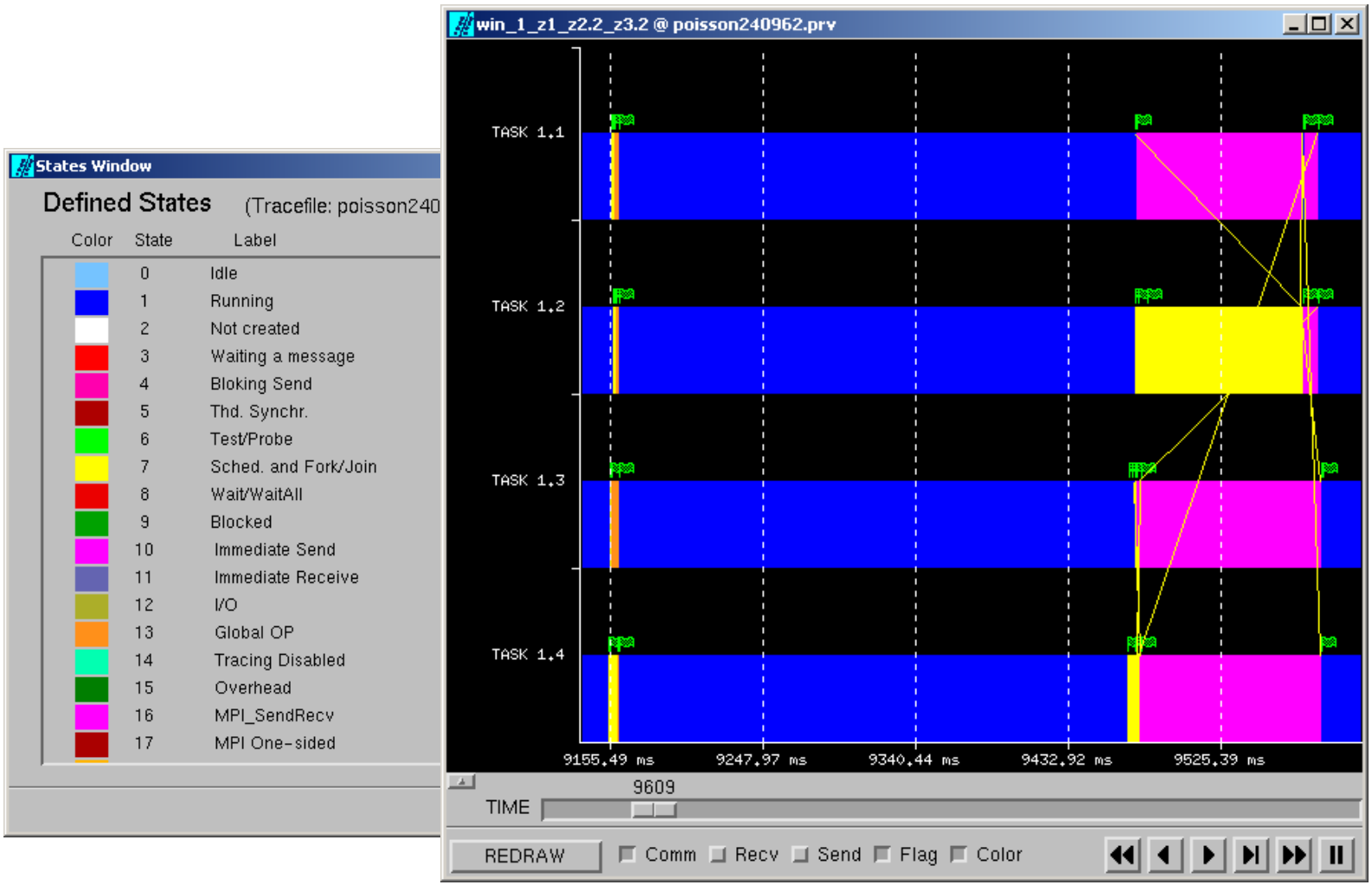


paraver

- If you graph something new, click on **`F`** in the Visualizer window, and then `Apply` to redraw with the correct scale.

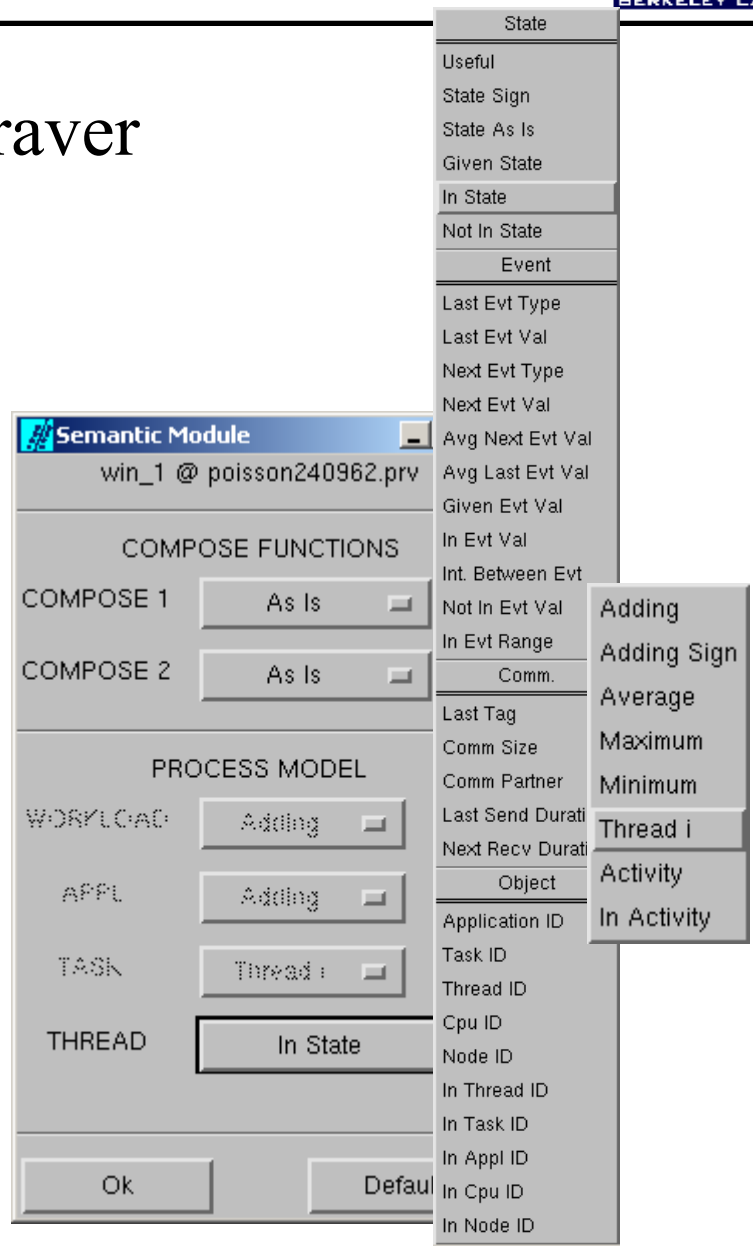


paraver


win_1_z1_z2.2_z3.2 @ poisson240962.prv
TASK 1.1
TASK 1.2
TASK 1.3
TASK 1.4
9155.49 ms
9247.97 ms
9340.44 ms
9432.92 ms
9525.39 ms
9609
TIME
REDRAW
Comm
Recv
Send
Flag
Color
States Window
Defined States
(Tracefile: poisson240
Color
State
Label
0 Idle
1 Running
2 Not created
3 Waiting a message
4 Bloking Send
5 Thd. Synchr.
6 Test/Probe
7 Sched. and Fork/Join
8 Wait/WaitAll
9 Blocked
10 Immediate Send
11 Immediate Receive
12 I/O
13 Global OP
14 Tracing Disabled
15 Overhead
16 MPI_SendRecv
17 MPI One-sided

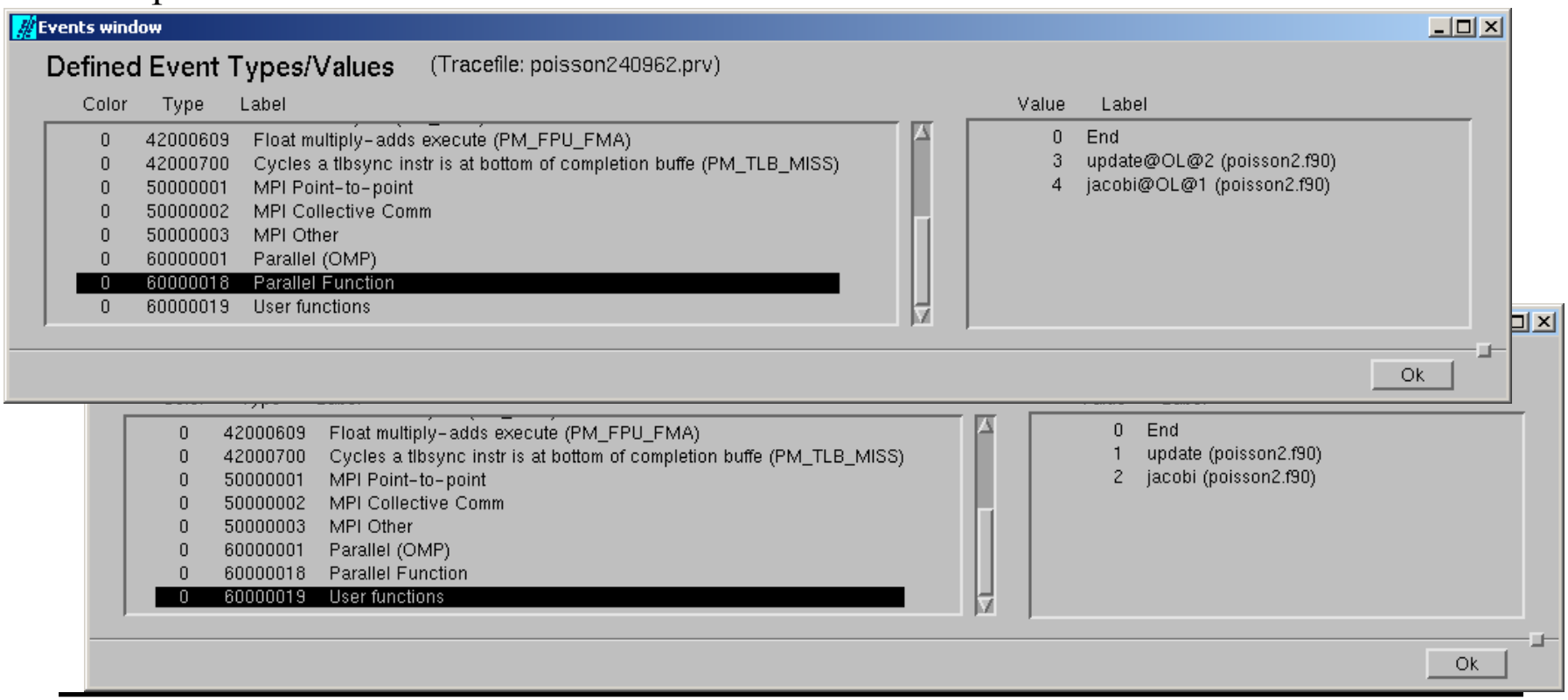
paraver

- Semantic Module has many options to
  - present State Values, Event Types, and Event States to the Visualizer Module
  - merge these values going from threads to tasks to applications



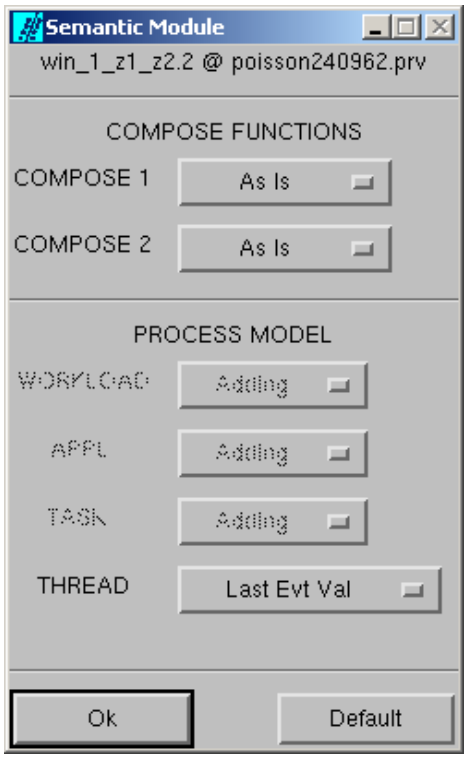
paraver

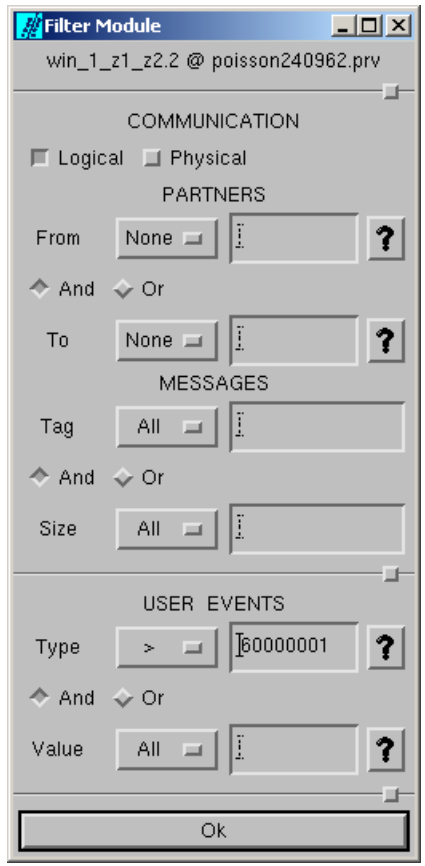
- Use Paraver window "Tracefiles->Trace Information->Defined Event Type/Values Window" to see all events recorded in the tracefile.
- Suppose we are interested in displaying the two main user functions and the OpenMP versions of those functions



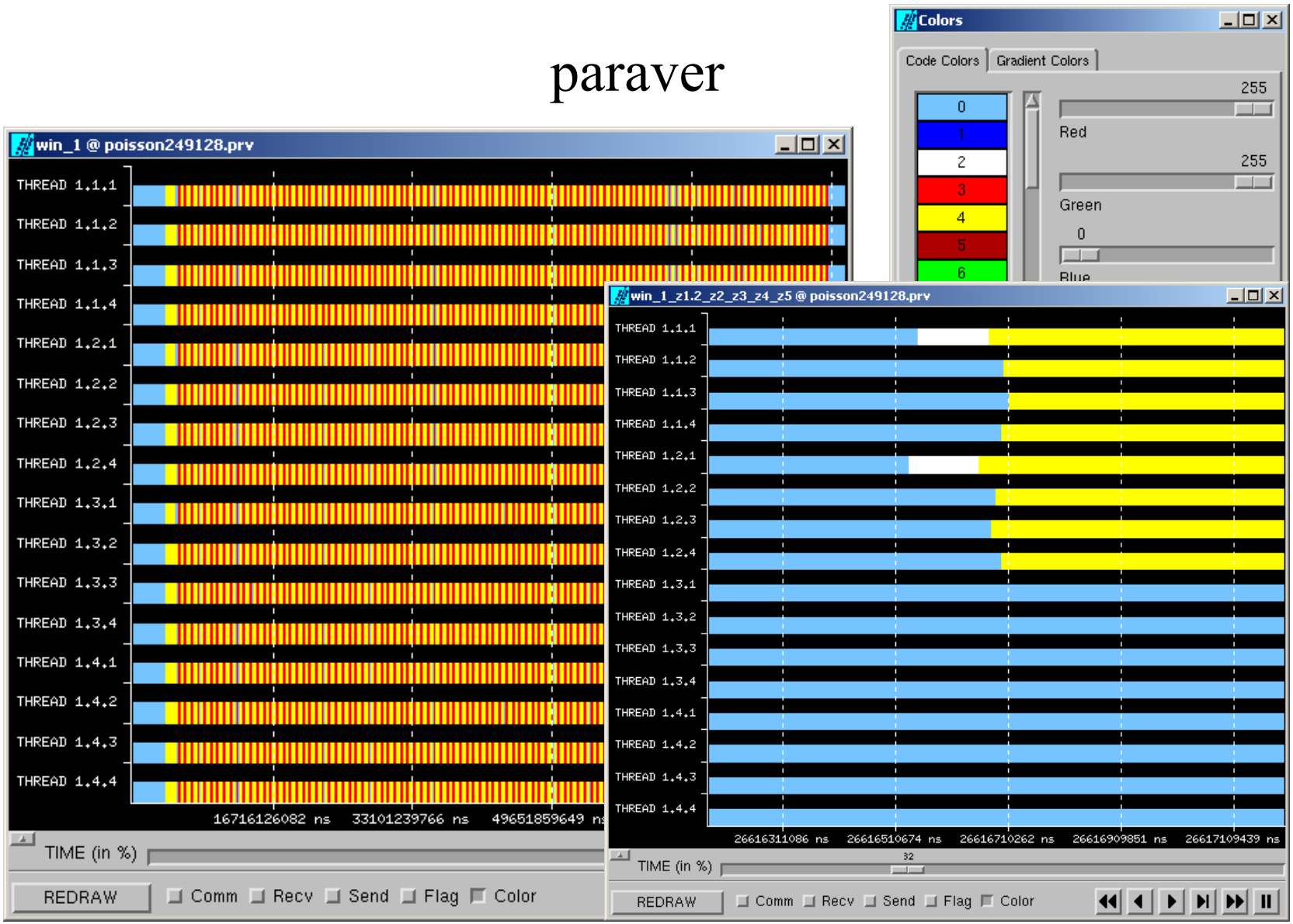
paraver

- Use Filter Module to remove all Communication Events, and all User Events not relevant to functions
- Use Semantic Module to display the Function value





paraver

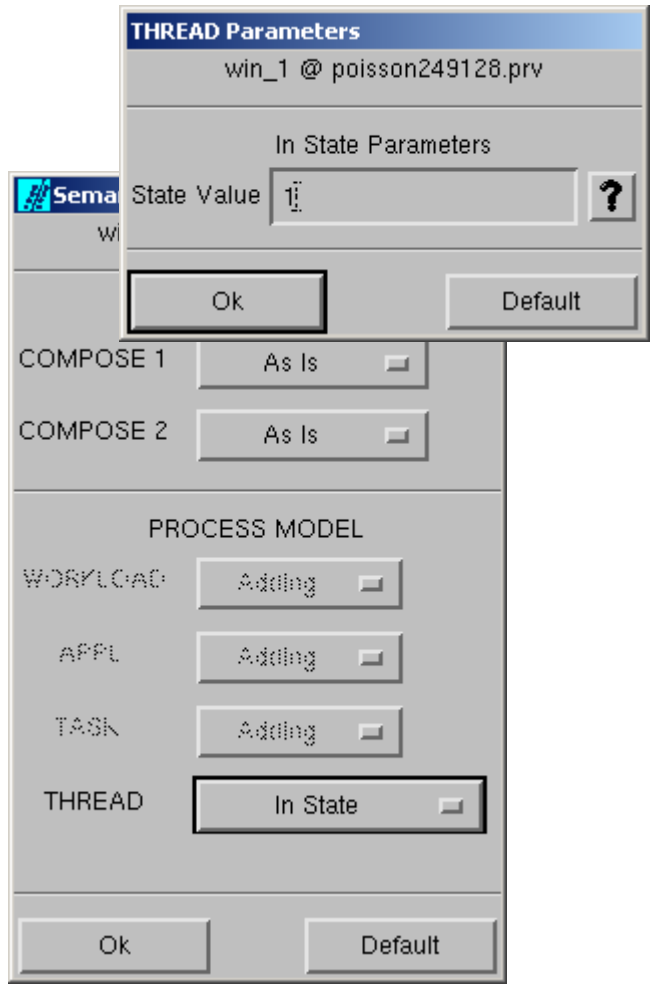

Colors
Code Colors
Gradient Colors
0
1
2
3
4
5
6
255
Red
255
Green
0
Blue
win_1 @ poisson249128.prv
THREAD 1.1.1
THREAD 1.1.2
THREAD 1.1.3
THREAD 1.1.4
THREAD 1.2.1
THREAD 1.2.2
THREAD 1.2.3
THREAD 1.2.4
THREAD 1.3.1
THREAD 1.3.2
THREAD 1.3.3
THREAD 1.3.4
THREAD 1.4.1
THREAD 1.4.2
THREAD 1.4.3
THREAD 1.4.4
16716126082 ns
33101239766 ns
49651859649 ns
TIME (in %)
REDRAW
Comm
Recv
Send
Flag
Color
win_1_z1.2_z2_z3_z4_z5 @ poisson249128.prv
THREAD 1.1.1
THREAD 1.1.2
THREAD 1.1.3
THREAD 1.1.4
THREAD 1.2.1
THREAD 1.2.2
THREAD 1.2.3
THREAD 1.2.4
THREAD 1.3.1
THREAD 1.3.2
THREAD 1.3.3
THREAD 1.3.4
THREAD 1.4.1
THREAD 1.4.2
THREAD 1.4.3
THREAD 1.4.4
26616311086 ns
26616510674 ns
26616710262 ns
26616909851 ns
26617109439 ns
32
TIME (in %)
REDRAW
Comm
Recv
Send
Flag
Color

paraver

- The 1D and 2D Analyzer modules can be used to evaluate statistics on state values and event values.
- The 1D Analyzer evaluates a set of functions applied to the output of the semantic module.
- The 2D Analyzer produces a matrix of values, paraver objects vs. states, events, or event values.

1D Analyzer example:

- How much time per thread is spent actually computing (not thread management, not MPI, not idle) ?
- Turn off all filters in Filter Module, set for thread-level Display window in Visualizer Module, use following Semantic Module settings:
- Click on 1D Analyzer button, then click on "All Trace" button in the new window



1D Analyzer

Analyzer: Analysis computed for win_1 @ poisson240962.prv

Row	Avg Semantic Val	# Sends	# Receives	# Events
THREAD 1.1.1	0.93	200	200	8211
THREAD 1.1.2	0.90	0	0	3600
THREAD 1.1.3	0.90	0	0	3600
THREAD 1.1.4	0.90	0	0	3600
THREAD 1.2.1	0.93	200	200	8211
THREAD 1.2.2	0.90	0	0	3600
THREAD 1.2.3	0.90	0	0	3600
THREAD 1.2.4	0.90	0	0	3600
THREAD 1.3.1	0.93	200	200	8211
THREAD 1.3.2	0.90	0	0	3600
THREAD 1.3.3	0.90	0	0	3600
THREAD 1.3.4	0.90	0	0	3600
THREAD 1.4.1	0.93	200	200	8211
THREAD 1.4.2	0.90	0	0	3600
THREAD 1.4.3	0.90	0	0	3600
THREAD 1.4.4	0.90	0	0	3600
Total	14.49	800	800	76044
Average	0.91	50	50	4752.75
Maximum	0.93	200	200	8211
Minimum	0.90	0	0	3600
	Graph/Text	Graph/Text	Graph/Text	Graph/Text

Ok
Save
Begin Time :
0.00 ms
End Time :
84732.34 ms
Duration :
84732.34 ms
All Window
All trace
Analyze
Repeat
Calculate All

Results show OpenMP threads are about 90% efficient, MPI threads about 93% efficient.

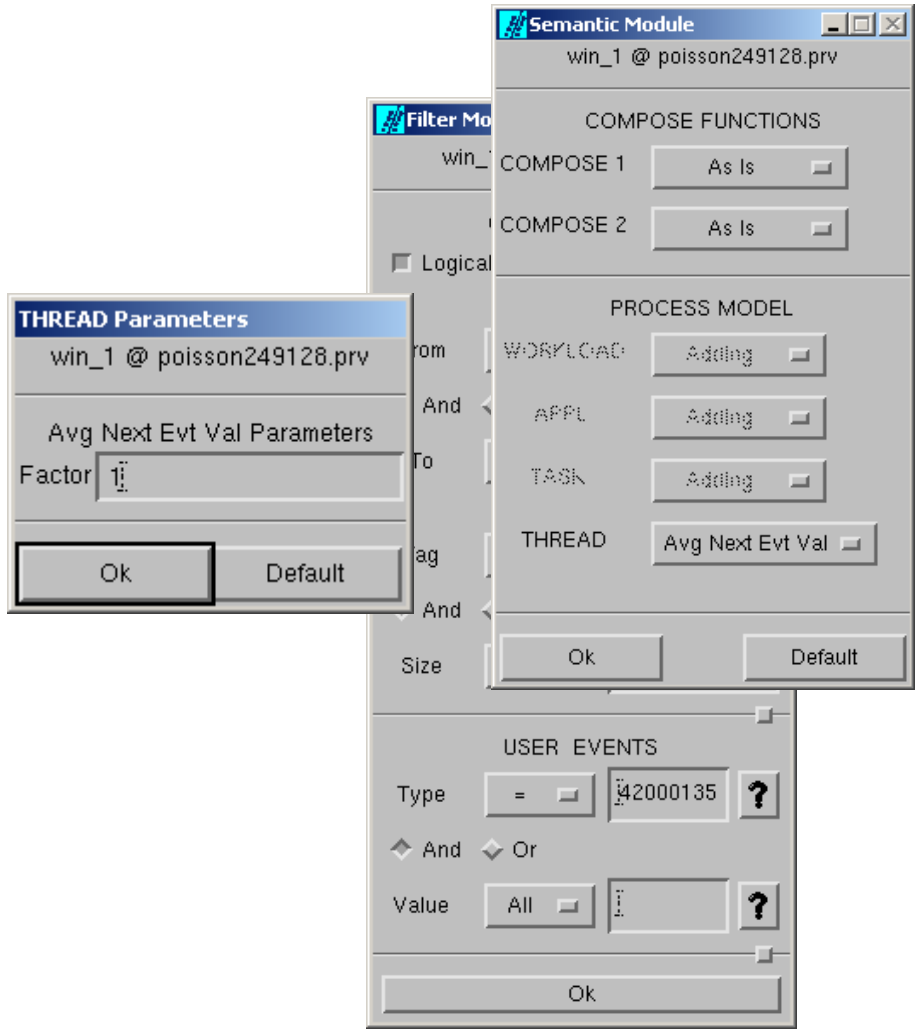
Derived Views

- Power 3 hardware counter values can be visualized over the whole run
- Mflop/s is given approximately by adding the values of the `PM_FPU0_CMPL, PM_FPU1_CMPL, PM_FPU_FMA` events
- Create three display windows each showing the values of one counter and then add them.

Displaying First Event

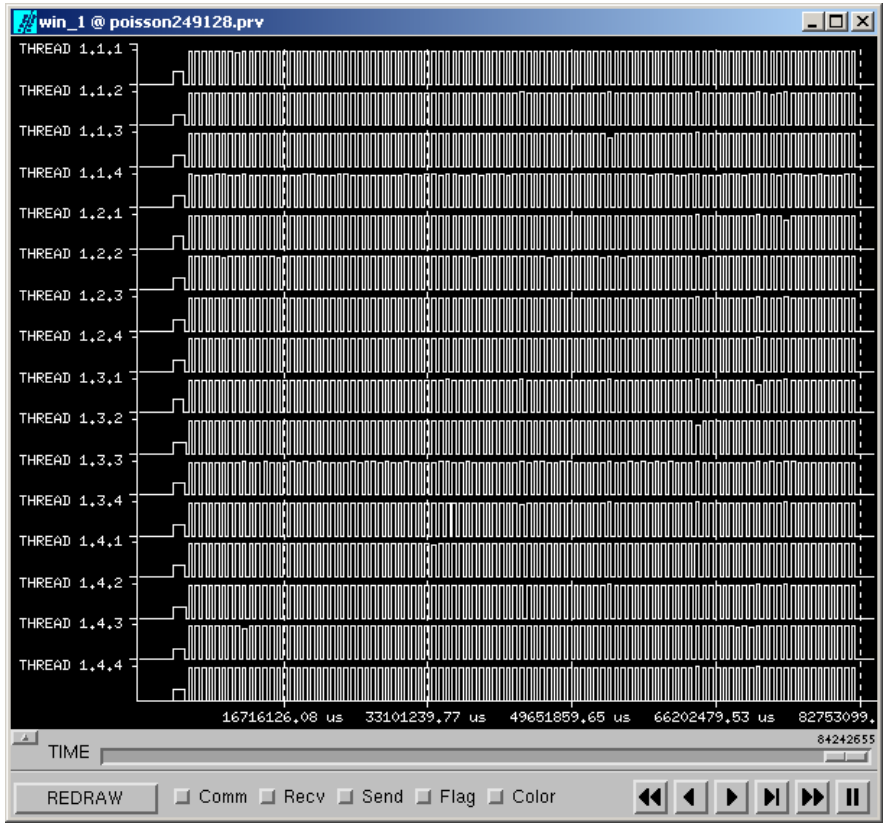
- `PM_FPU1_CMPL` is event 42000135, so filter for this event only

- Choose "Avg Next Evt Val" for the thread Semantic Module
  - Computes last value minus first value divided by time multiplied by "Factor"
  - Units of time are indicated in Visualizer Module



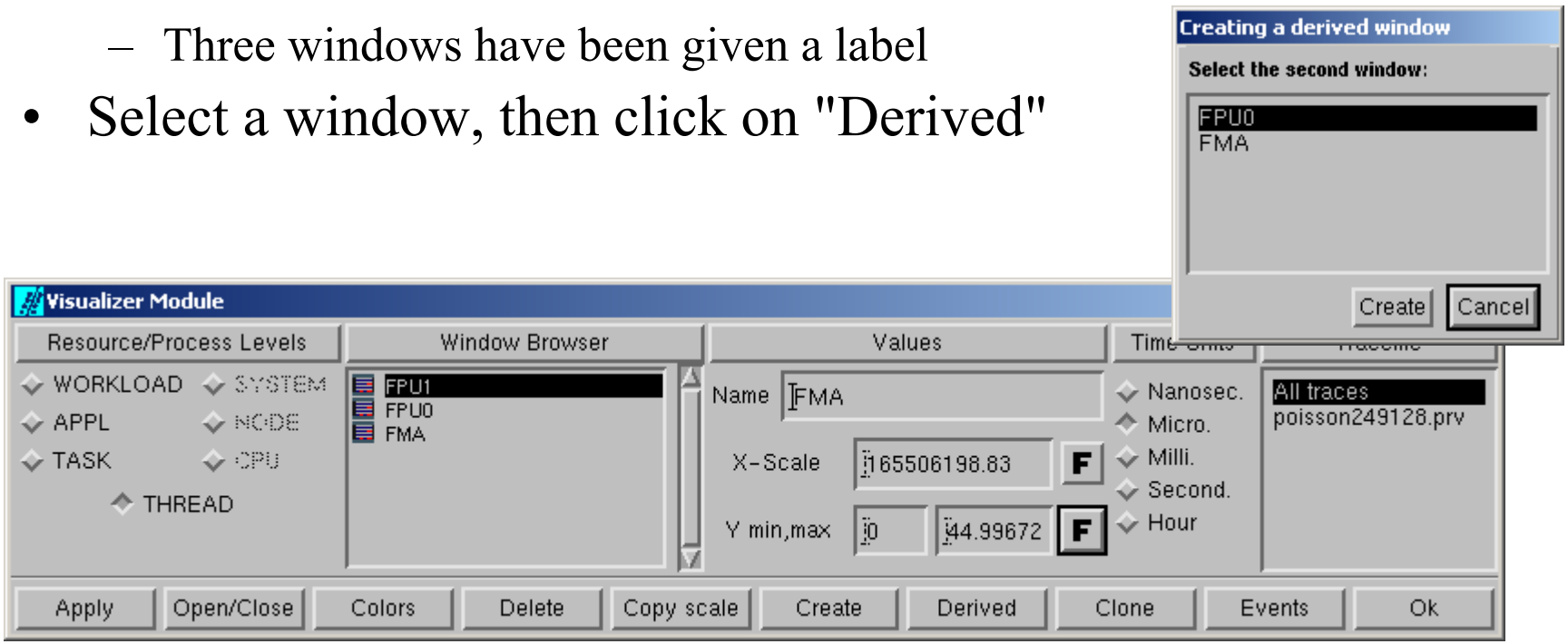
Displaying First Event

- Clicking on the "Color" checkbox plots a graph instead of color coding the value.
- Repeat this procedure with `PM_FPU0_CMPL` and `PM_FPU_FMA`



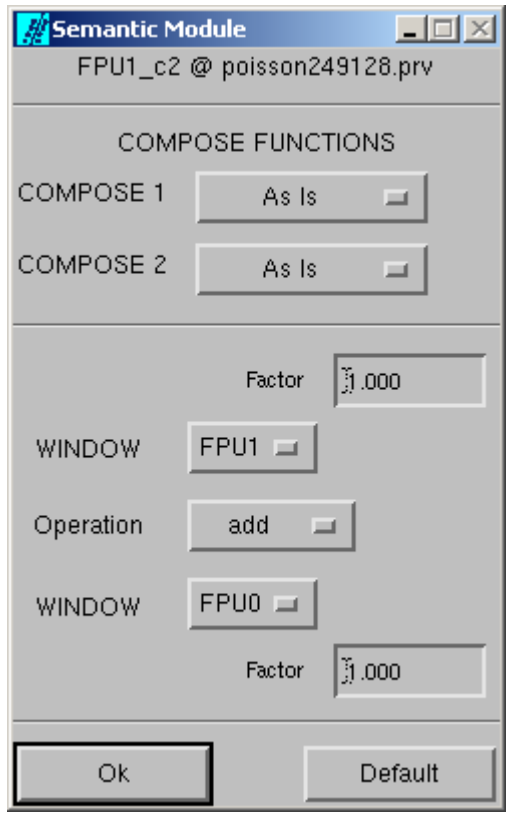
Creating Derived Window

- Visualizer Module should look like this:
  - Three windows have been given a label
- Select a window, then click on "Derived"



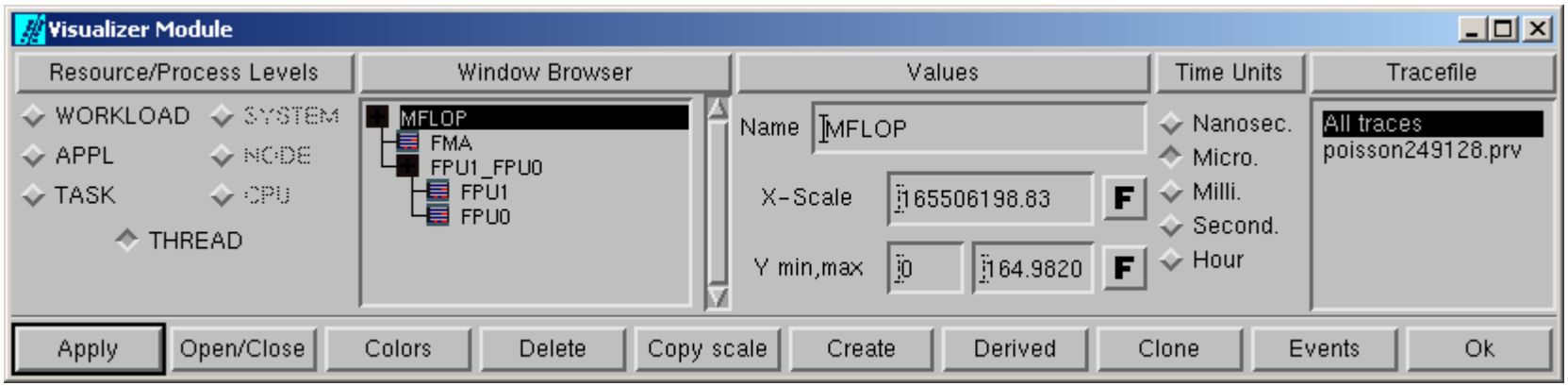
Creating Derived Window

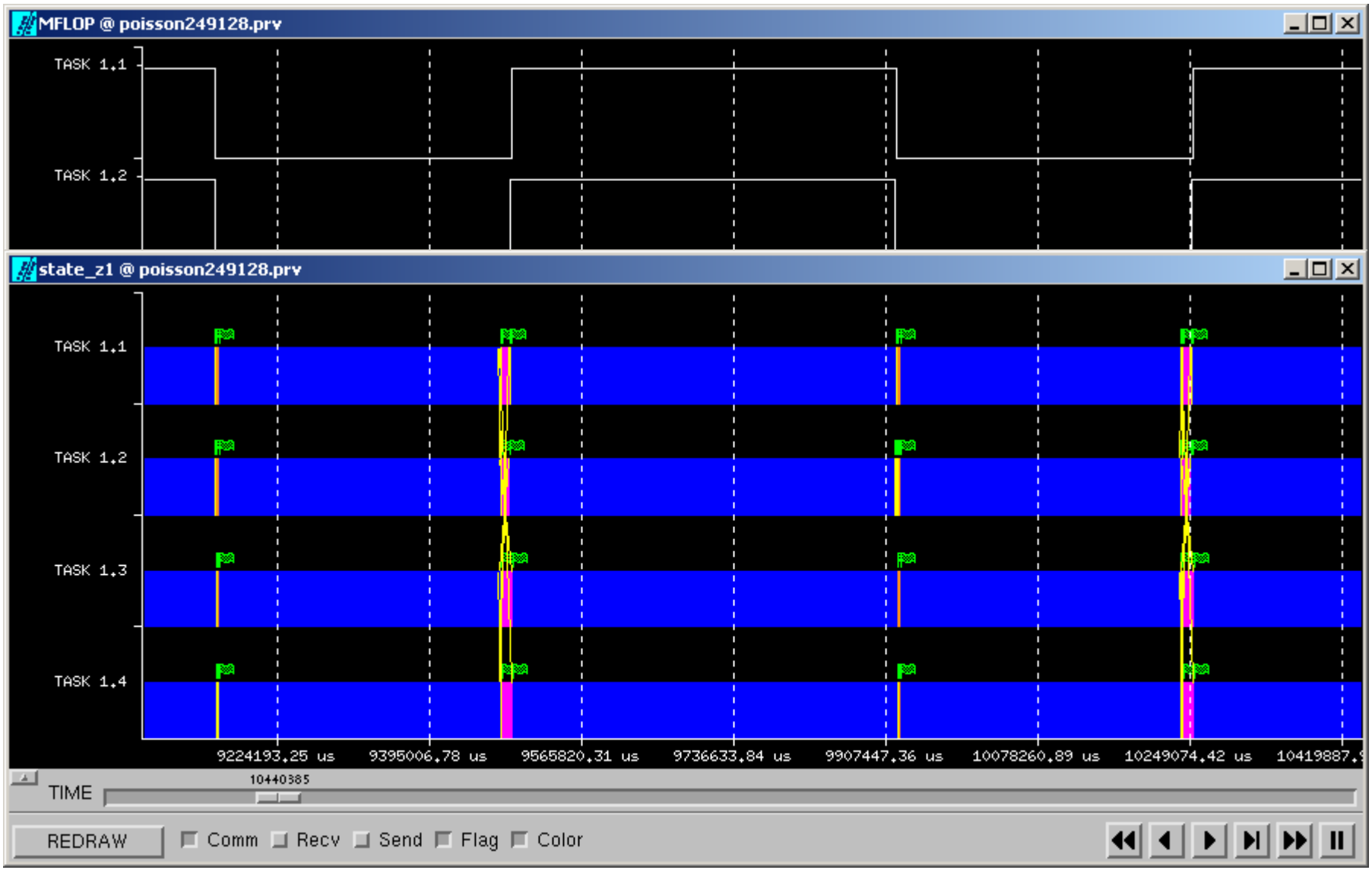
- Semantic Module window changes during creation of a derived view
- Select "add" as the Operation to combine the two windows. Default is "product".



Creating Derived Window

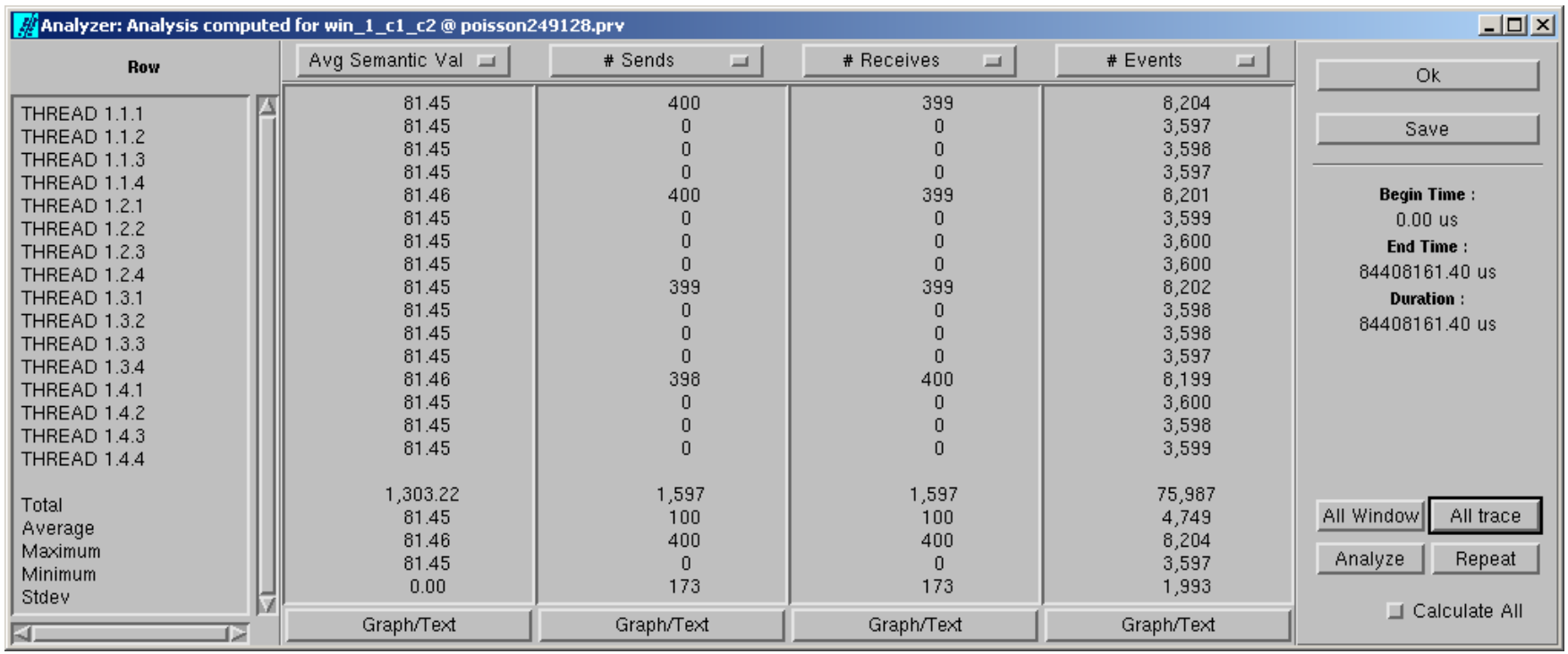
- Add FMA window to FPU0+FPU1 window




MFLOP @ poisson249128.prv
TASK 1.1
TASK 1.2
state_z1 @ poisson249128.prv
TASK 1.1
TASK 1.2
TASK 1.3
TASK 1.4
9224193.25 us
9395006.78 us
9565820.31 us
9736633.84 us
9907447.36 us
10078260.89 us
10249074.42 us
TIME
10440385
REDRAW
Comm
Recv
Send
Flag
Color

Mflop/s 1D Analyzer View

Using the 1D Analyzer on the Mflop window produces the same kind of output as hpmcount.



Further Reading

- CEPBA Tools
  - http://www.cepba.upc.es/paraver/index.html
  - Not completely updated for IBM SP

おわり