

Projections Overview

Ronak Buch & Laxmikant (Sanjay) Kale

<http://charm.cs.illinois.edu>

Parallel Programming Laboratory

Department of Computer Science

University of Illinois at Urbana-Champaign

Manual

<http://charm.cs.illinois.edu/manuals/html/projections/manual-1p.html>

Full reference for Projections, contains more details than these slides.

Projections

- Performance analysis/visualization tool for use with Charm++
 - Works to limited degree with MPI
- Charm++ uses runtime system to log execution of programs
- Trace-based, post-mortem analysis
- Configurable levels of detail
- Java-based visualization tool for performance analysis

Instrumentation

- Enabling Instrumentation
- Basics
- Customizing Tracing
- Tracing Options

How to Instrument Code

- Build Charm++ with the `--enable-tracing` flag
- Select a `-tracemode` when linking
- That's all!
- Runtime system takes care of tracking events

Basics

Traces include variety of events:

- Entry methods
 - Methods that can be remotely invoked
- Messages sent and received
- System Events
 - Idleness
 - Message queue times
 - Message pack times
 - etc.

Basics - Continued

- Traces logged in memory and incrementally written to disk
- Runtime system instruments computation and communication
- Generates useful data without excessive overhead (usually)

Custom Tracing - User Events

Users can add custom events to traces by inserting calls into their application.

Register Event:

```
int traceRegisterUserEvent(char* EventDesc, int  
EventNum=-1)
```

Track a Point-Event:

```
void traceUserEvent(int EventNum)
```

Track a Bracketed-Event:

```
void traceUserBracketEvent(int EventNum, double  
StartTime, double EndTime)
```


Custom Tracing - Annotations

Annotation supports allows users to easily customize the set of methods that are traced.

- Annotating entry method with `notrace` avoids tracing and saves overhead
- Adding `local` to non-entry methods (not traced by default) adds tracing automatically

Custom Tracing - API

API allows users to turn tracing on or off:

- Trace only at certain times
- Trace only subset of processors

Simple API:

- `void traceBegin()`
- `void traceEnd()`

Works at granularity of PE.

Custom Tracing - API

- Often used at synchronization points to only instrument a few iterations
- Reduces size of logs while still capturing important data
- Allows analysis to be focused on only certain parts of the application

Tracing Options

Two link-time options:

- tracemode projections

 - Full tracing (time, sending/receiving processor, method, object, ...)

- tracemode summary

 - Performance of each PE aggregated into time bins of equal size

Tradeoff between detail and overhead

Tracing Options - Runtime

- `+traceoff` disables tracing until a `traceBegin()` API call.
- `+traceroot <dir>` specifies output folder for tracing data
- `+traceprocessors RANGE` only traces PEs in RANGE

Tracing Options - Summary

- `+sumdetail` aggregate data by entry method as well as time-intervals. (normal summary data is aggregated only by time-interval)
- `+numbins <k>` reserves enough memory to hold information for `<k>` time intervals. (default is 10,000 bins)
- `+binsize <duration>` aggregates data such that each time-interval represents `<duration>` seconds of execution time. (default is 1ms)

Tracing Options - Projections

- `+logsize <k>` reserves enough buffer memory to hold `<k>` events. (default is 1,000,000 events)
- `+gz-trace`, `+gz-no-trace` enable/disable compressed (gzip) log files

Memory Usage

What happens when we run out of reserved memory?

- `-tracemode summary`: doubles time-interval represented by each bin, aggregates data into the first half and continues.
- `-tracemode projections`: asynchronously flushes event log to disk and continues. This can perturb performance significantly in some cases.

Projections Client

- Scalable tool to analyze up to 300,000 log files
- A rich set of tool features : time profile, time lines, usage profile, histogram, extrema tool
- Detect performance problems: load imbalance, grain size, communication bottleneck, etc
- Multi-threaded, optimized for memory efficiency

Visualizations and Tools

- Tools of aggregated performance viewing
 - Time profile
 - Histogram
 - Communication
- Tools of processor level granularity
 - Overview
 - Timeline
- Tools of derived/processed data
 - Outlier analysis: identifies outliers

Analysis at Scale

- Fine grain details can sometimes look like one big solid block on timeline.
- It is hard to mouse-over items that represent fine-grained events.
- Other times, tiny slivers of activity become too small to be drawn.

Analysis Techniques

- Zoom in/out to find potential problem spots.
- Mouseover graphs for extra details.
- Load sufficient but not too much data.
- Set colors to highlight trends.
- Use the history feature in dialog boxes to track time-ranges explored.

Select Range For Timeline ✕

Valid Processors = 0-7
Processors :

Valid Time Range = 0 to 315.89ms
Start Time : End Time :
Total Time selected : 100ms

Save History to Disk

Add to History List Remove selected History

Find annotated timesteps

Filter out entries shorter than

Filter out messages

Filter out user events

Highlight the top longest idle and entry times

OK Cancel

Dialog Box

Select processors: 0-2,4-7:2 gives 0,1,2,4,6

Select Range For Timeline

Valid Processors = 0-7

Processors : 0-2,4-7:2

Valid Time Range = 0 to 315.89ms

Start Time : 100ms End Time : 200ms

Total Time selected : 100ms

Add to History List Save History to Disk

Remove selected History

Find annotated timesteps

Filter out entries shorter than 0.03ms

Filter out messages

Filter out user events

Highlight the top 10 longest idle and entry times

OK Cancel

Dialog Box

Select time range

Select Range For Timeline

Valid Processors = 0-7

Processors : 0-2,4-7:2

Valid Time Range = 0 to 315.89ms

Start Time : 100ms End Time : 200ms

Total Time selected : 100ms

Save History to Disk

Add to History List Remove selected History

Find annotated timesteps

Filter out entries shorter than 0.03ms

Filter out messages

Filter out user events

Highlight the top 10 longest idle and entry times

OK Cancel

Dialog Box

Add presets to history

Select Range For Timeline

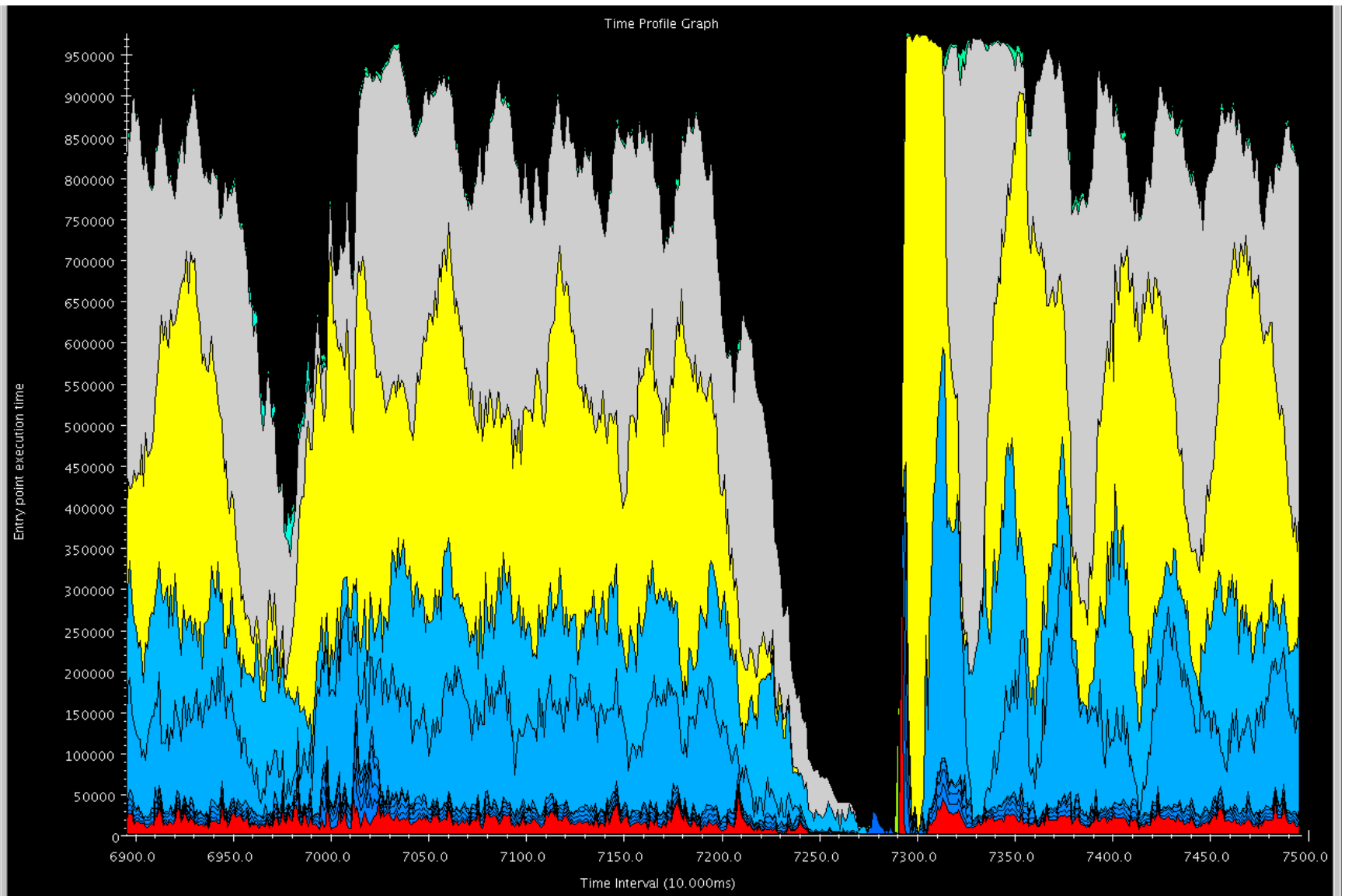
Valid Processors = 0-7
Processors : 0-2,4-7:2

Valid Time Range = 0 to 315.89ms
Start Time : 100ms End Time : 200ms
Total Time selected : 100ms

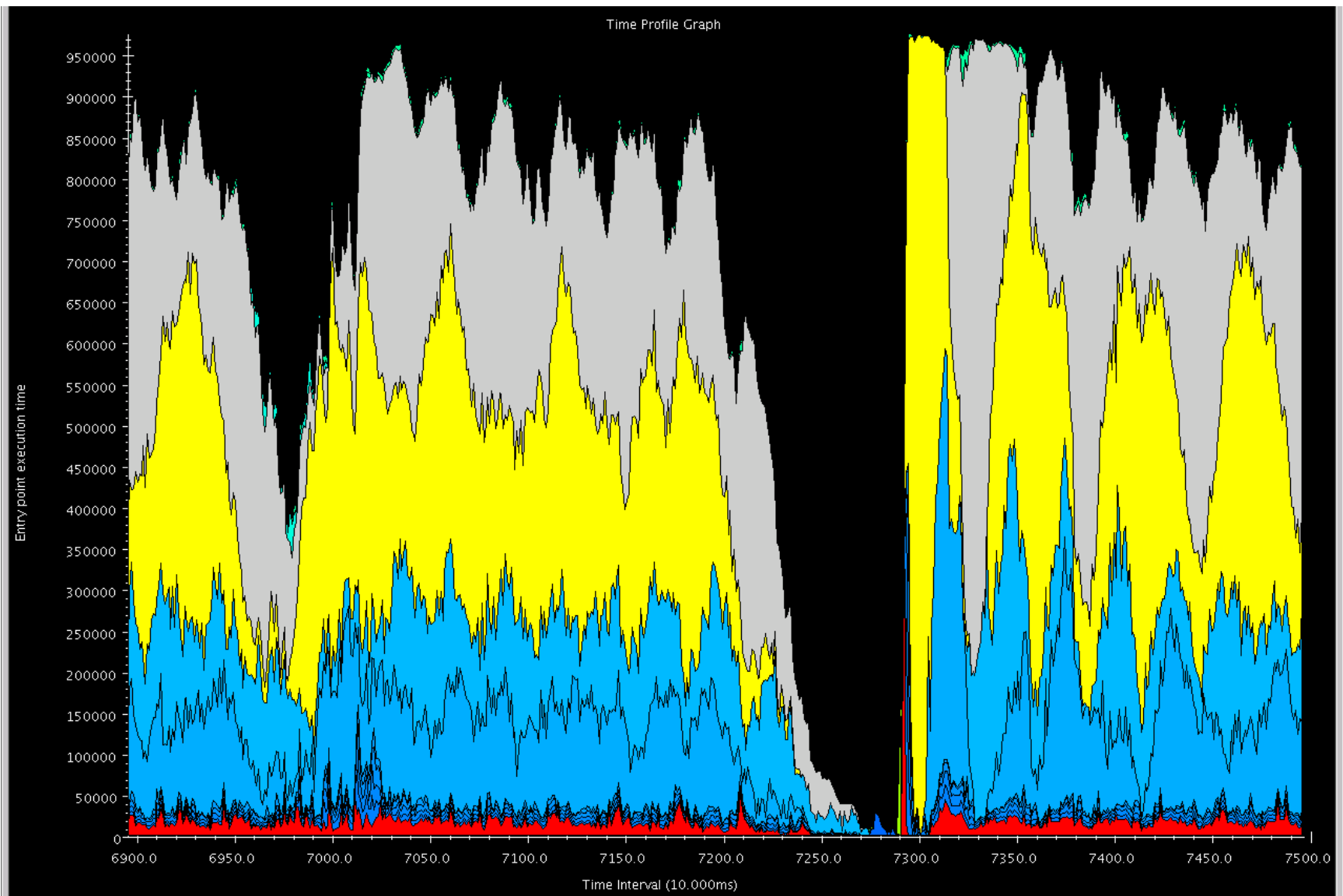
Filter out entries shorter than 0.03ms
 Filter out messages
 Filter out user events
 Highlight the top 10 longest idle and entry times

Dialog Box

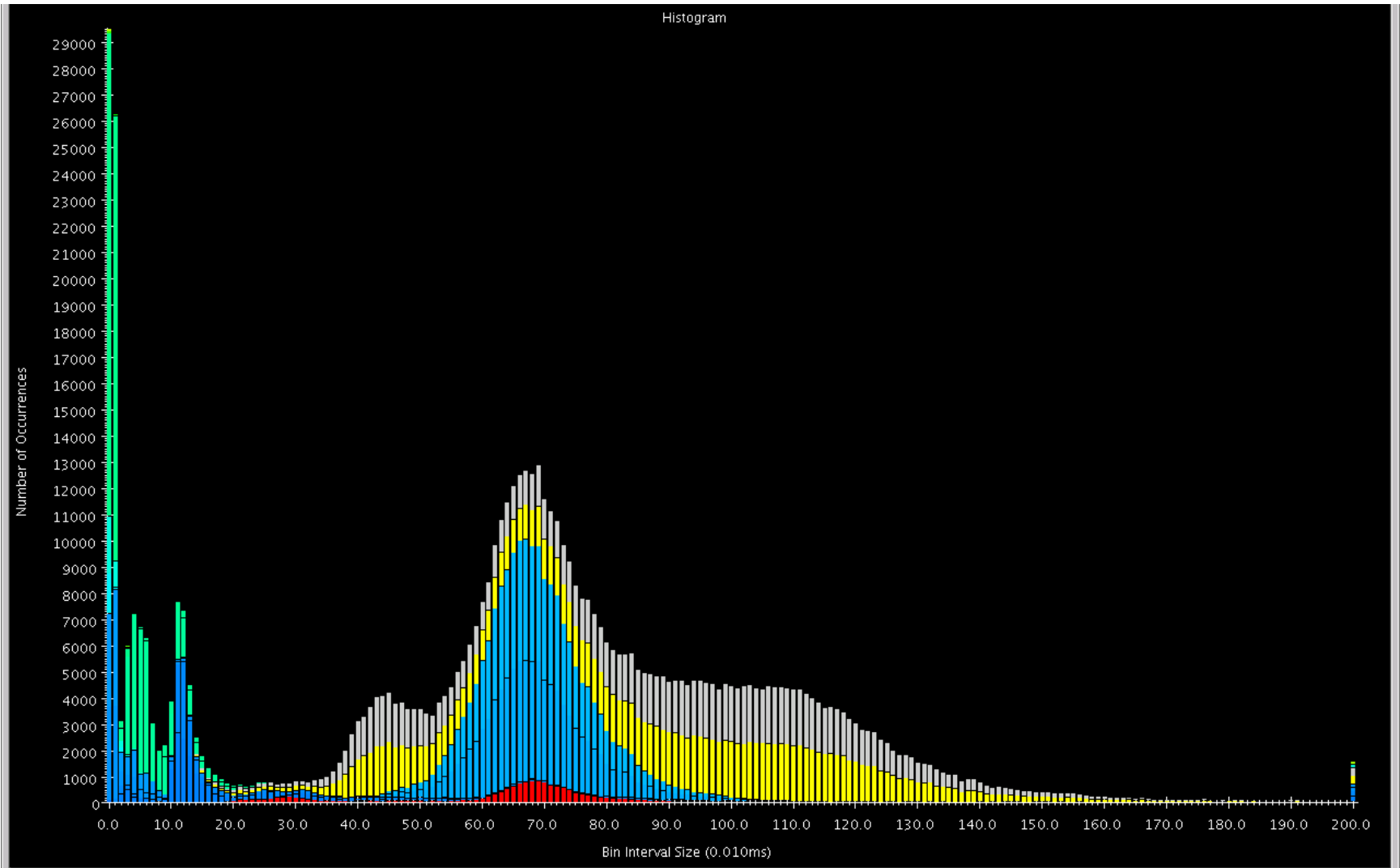
Aggregate Views



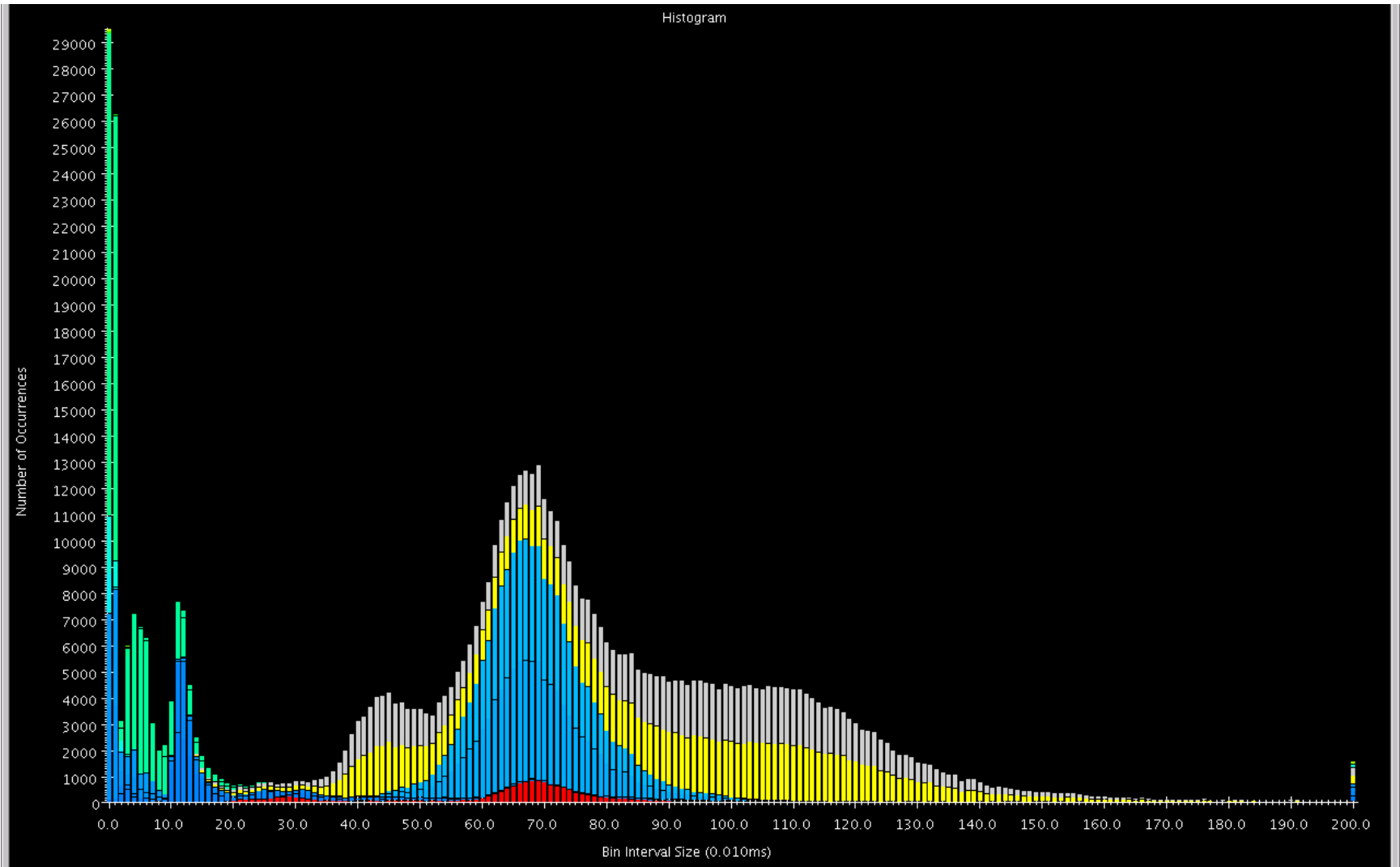
Time Profile



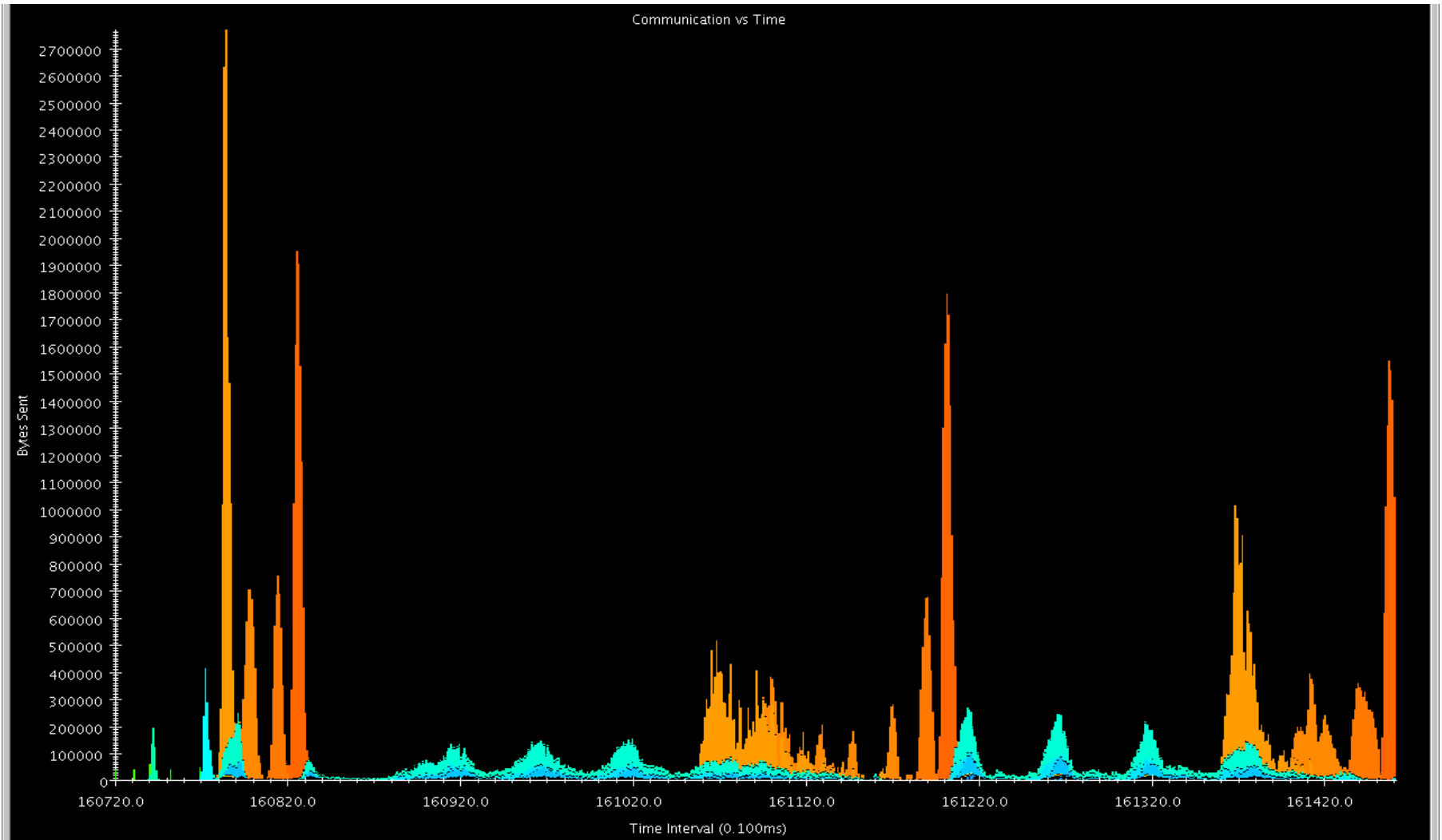
Time spent by each EP summed
across all PEs in time interval



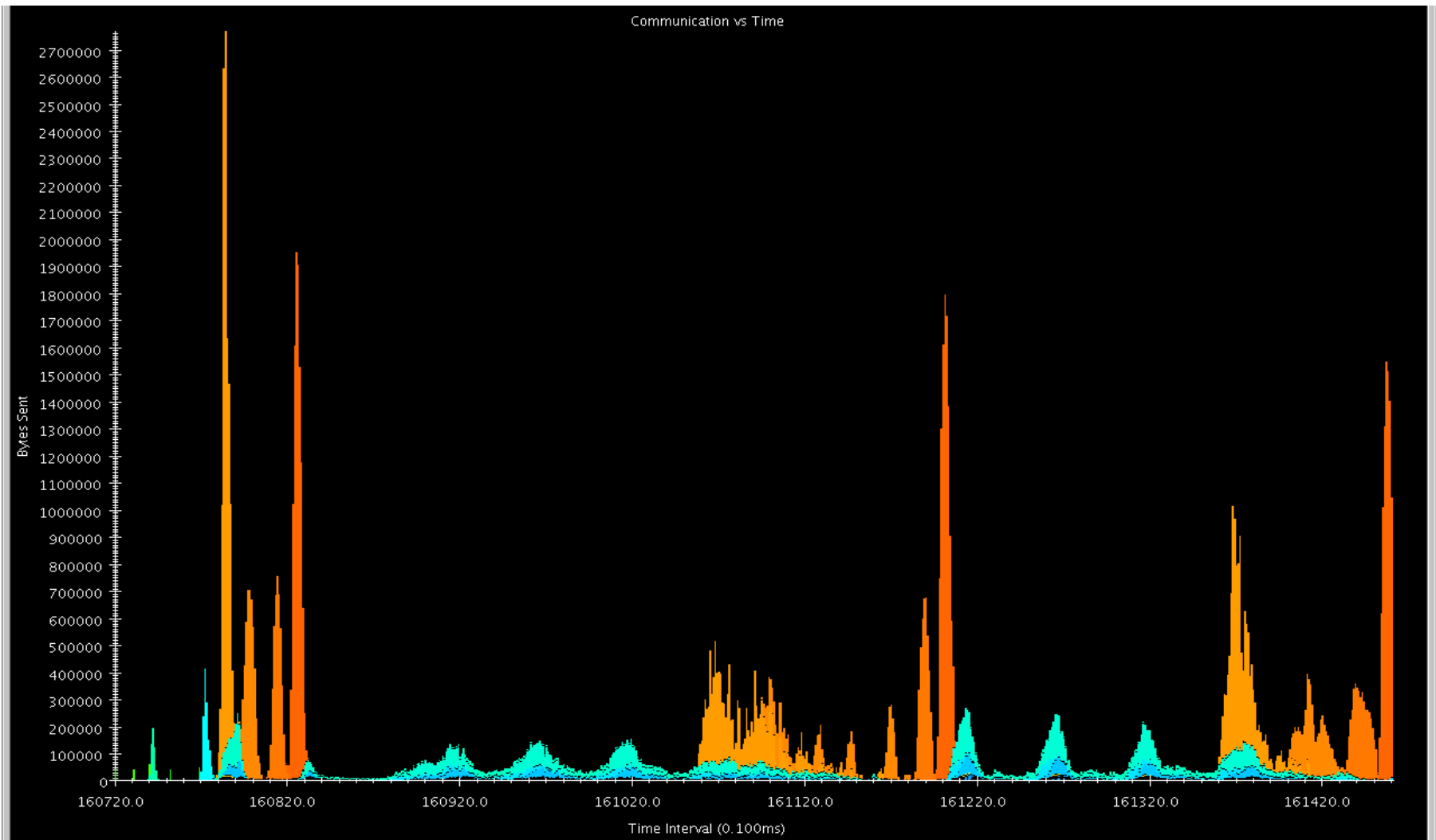
Histogram



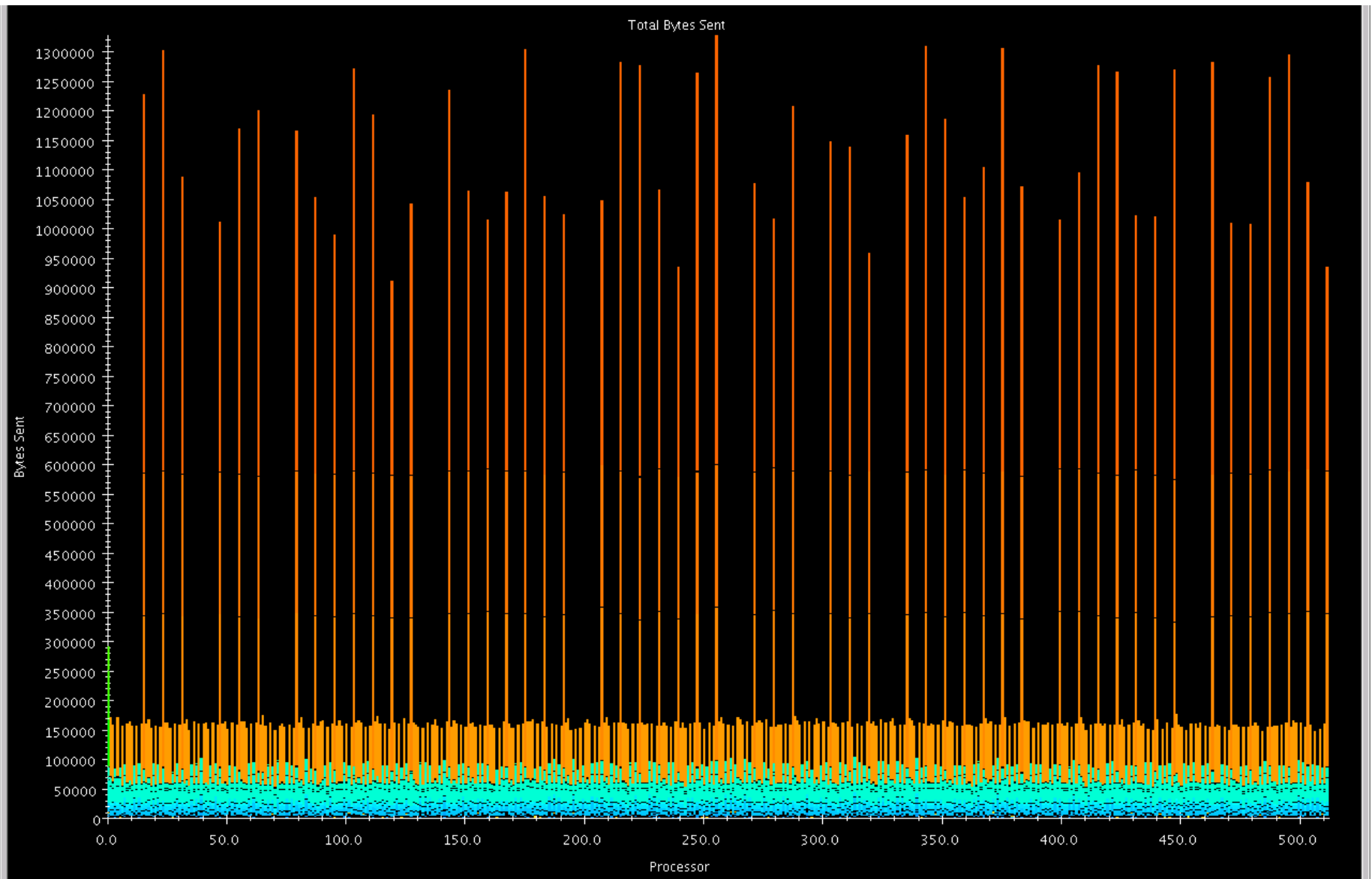
Shows statistics in “frequency” domain.



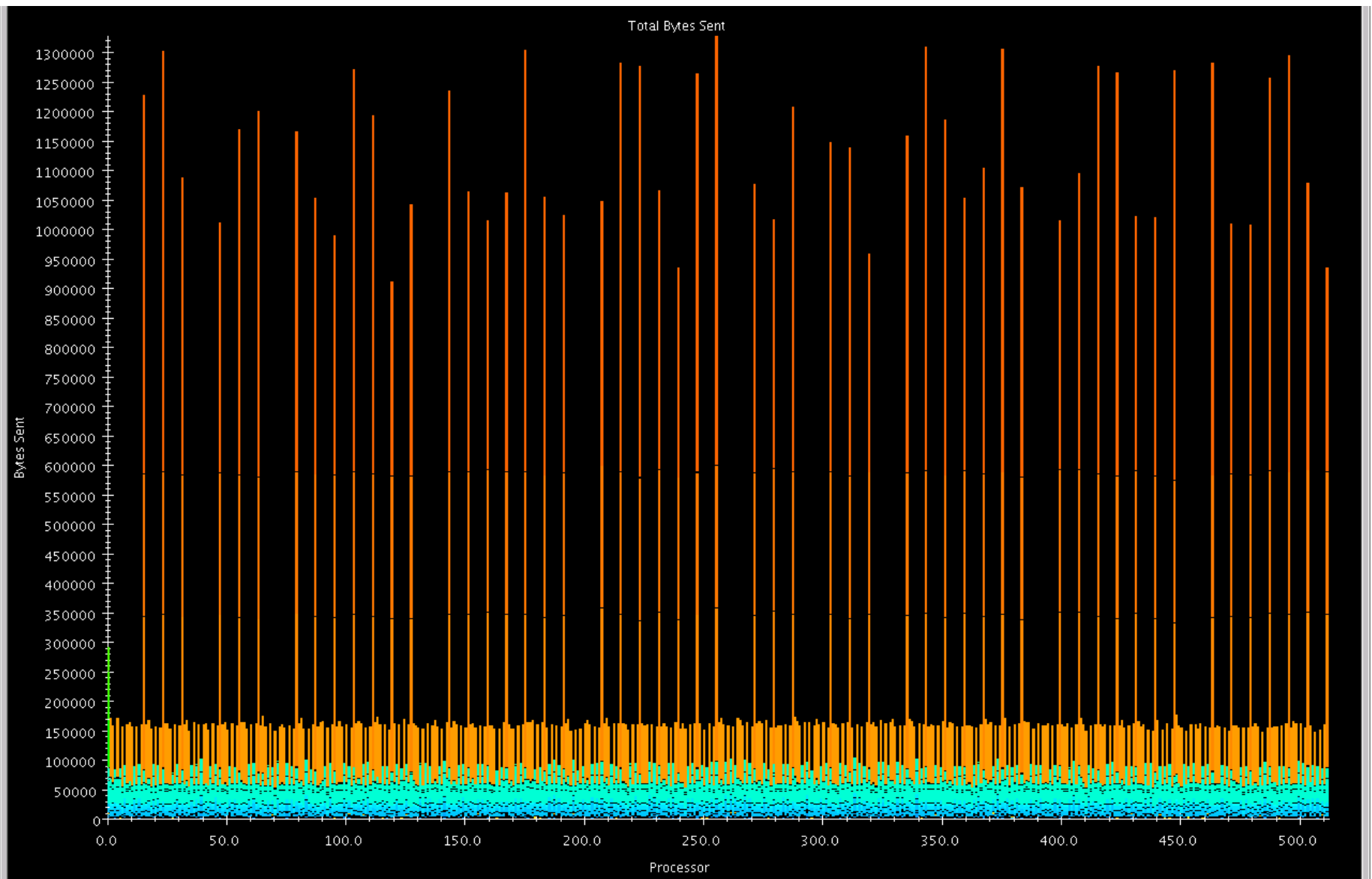
Communication vs. Time



Shows communication over all PEs in the time domain.

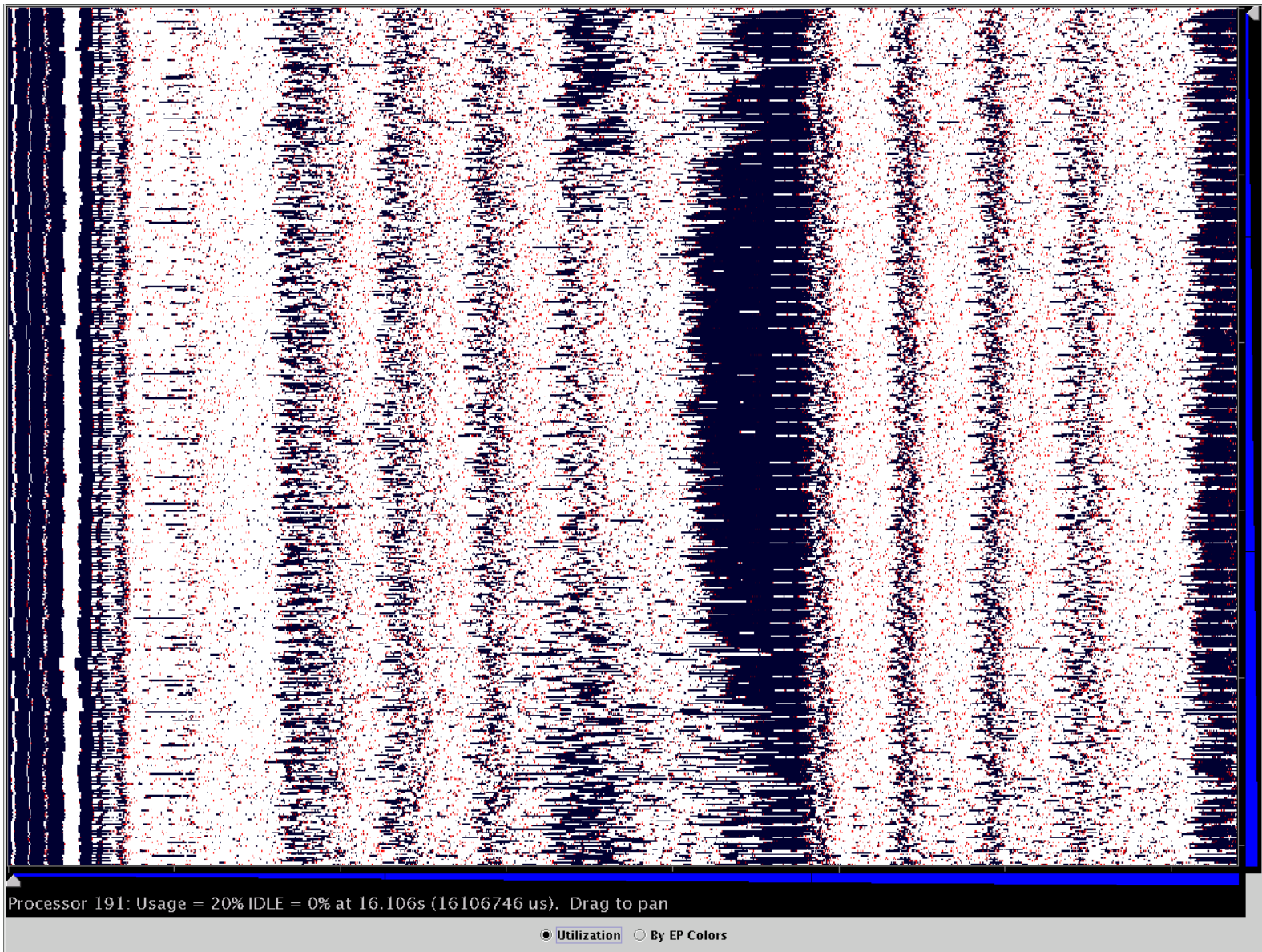


Communication per Processor

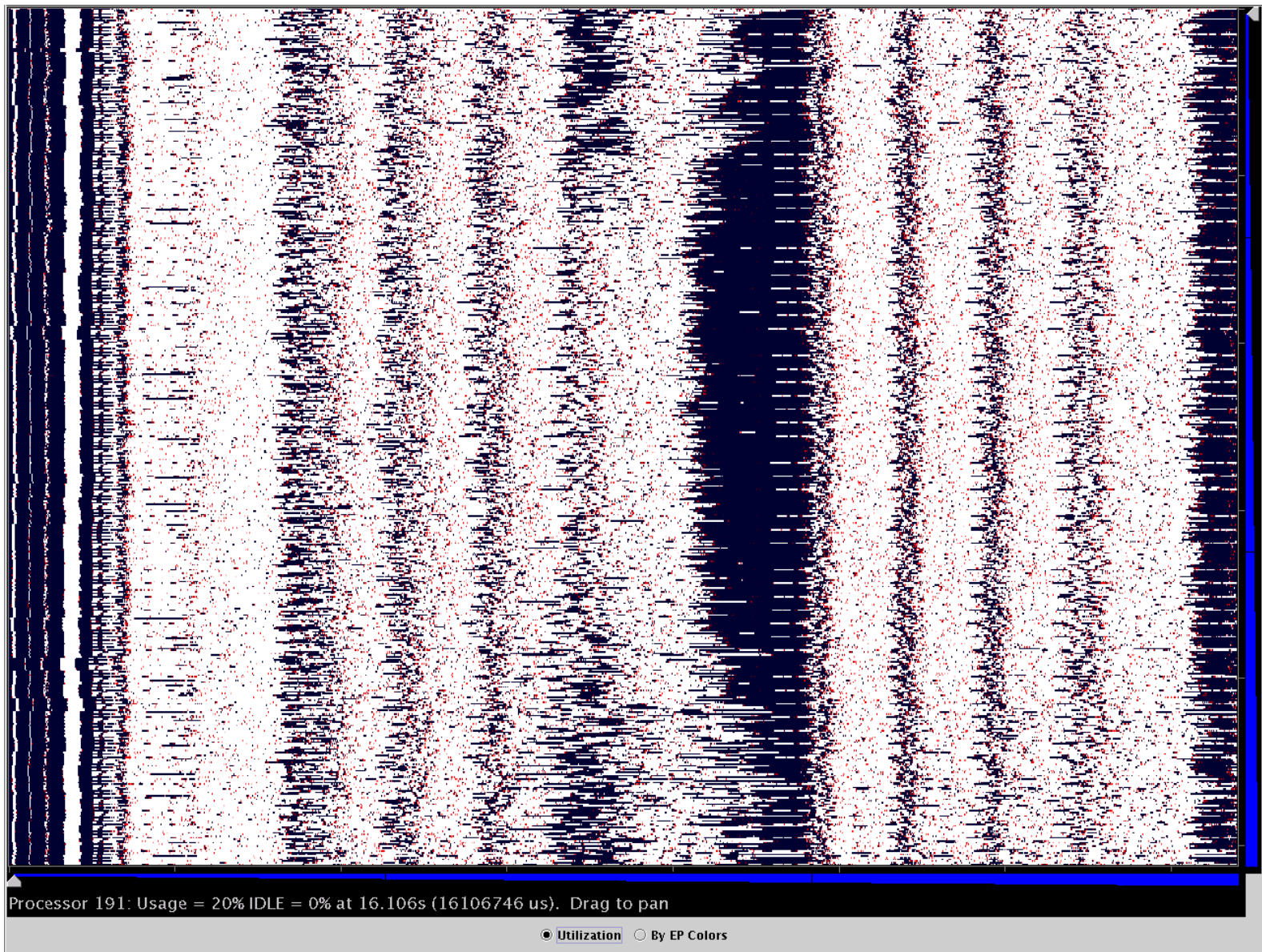


Shows how much each PE communicated over the whole job.

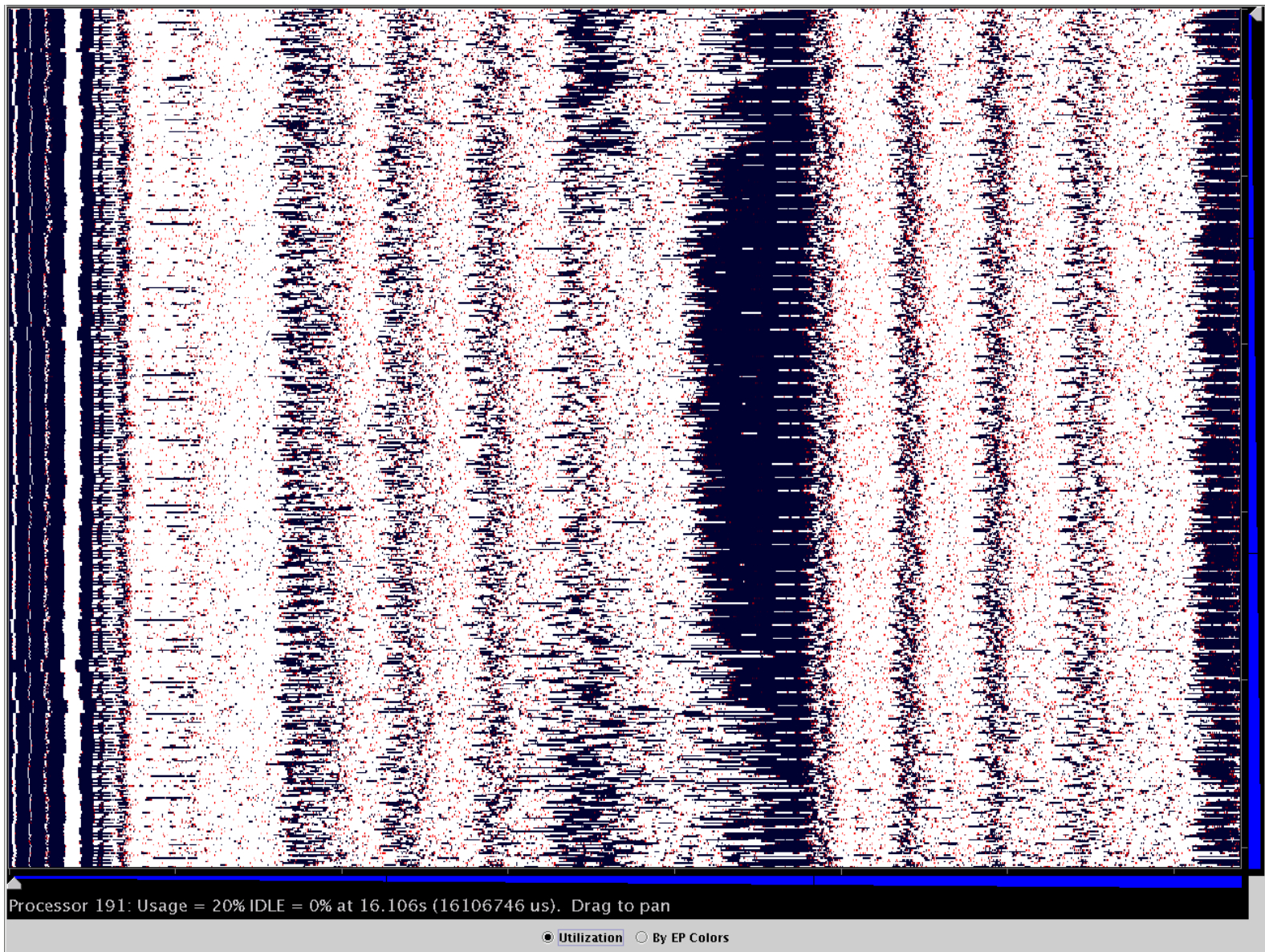
Processor Level Views



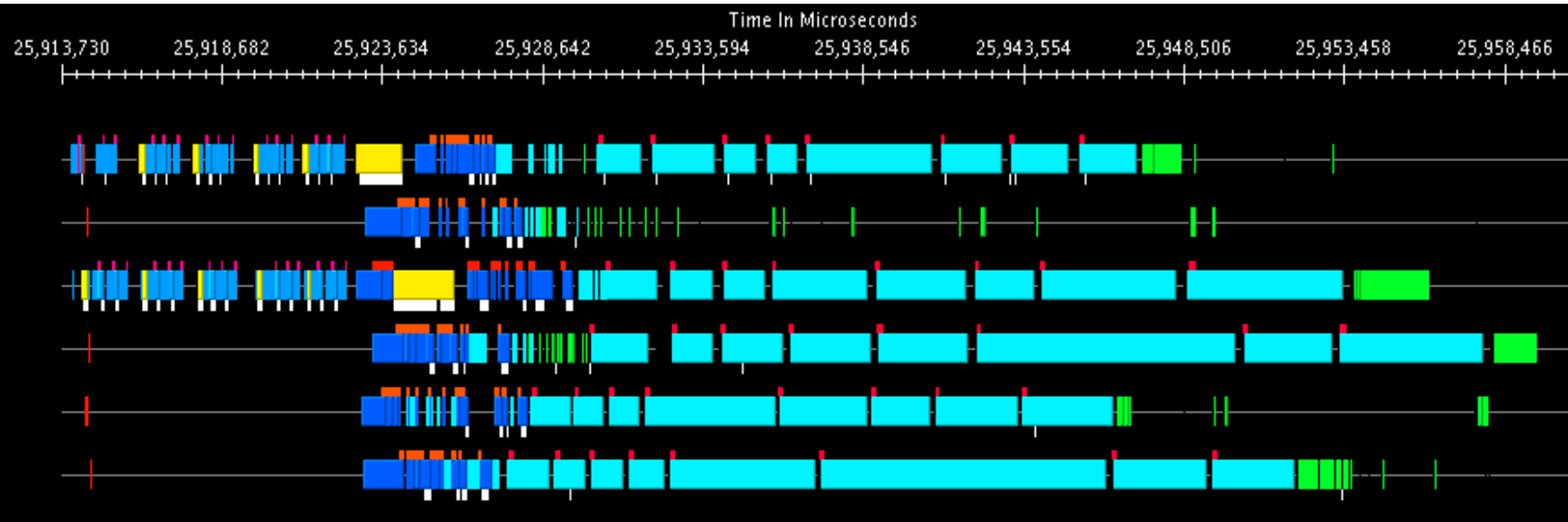
Overview



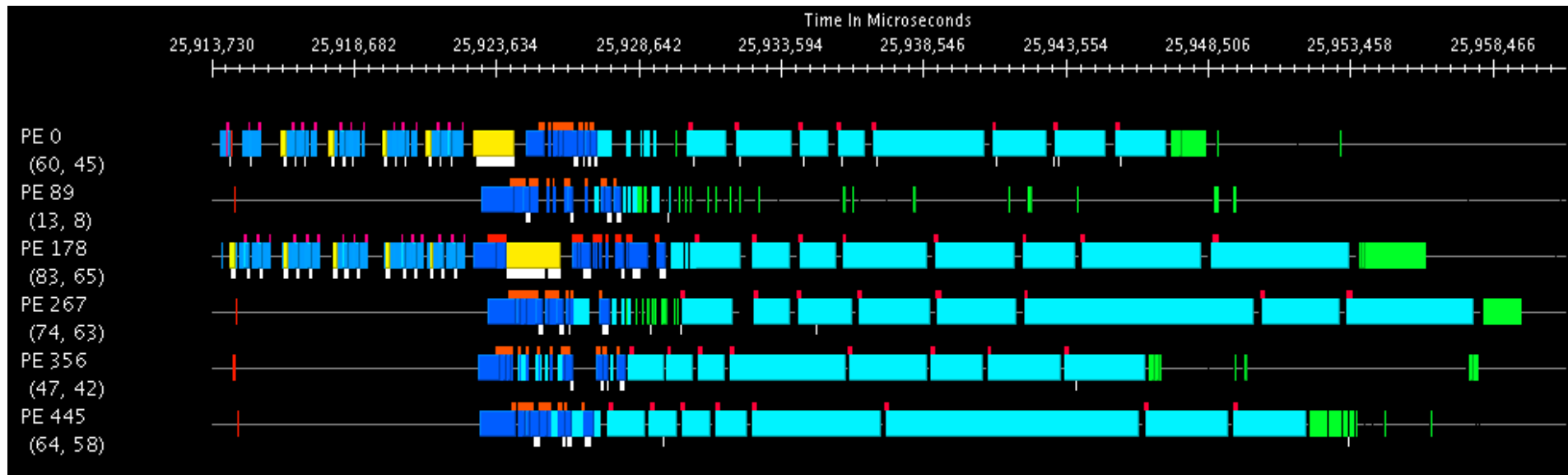
Time on X, different PEs on Y



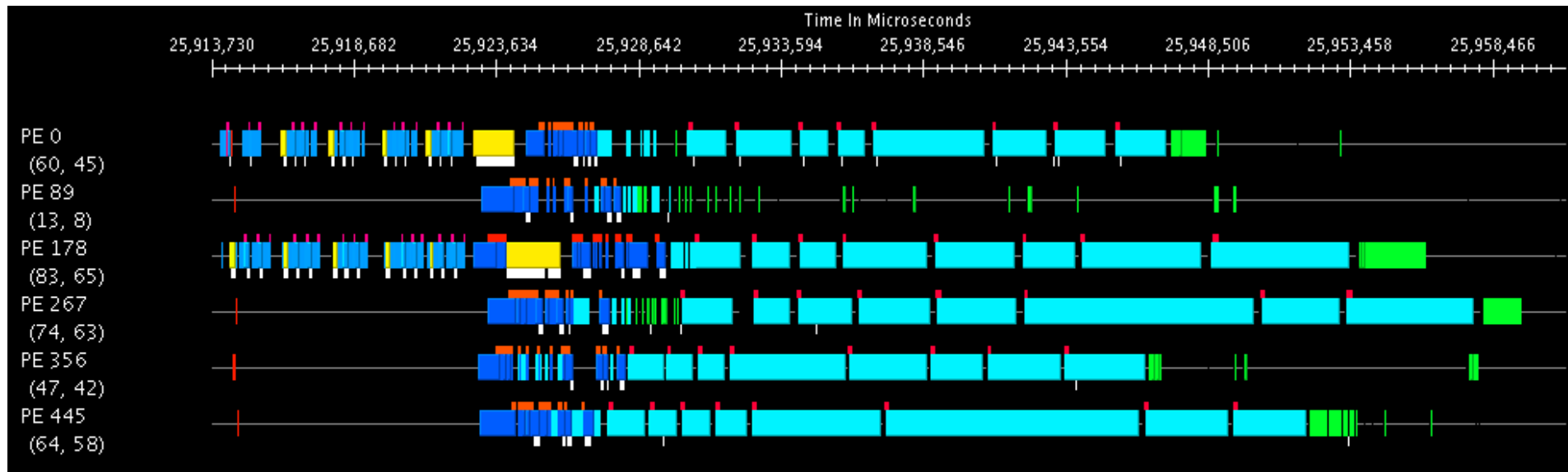
Intensity of plot represents PE's utilization at that time



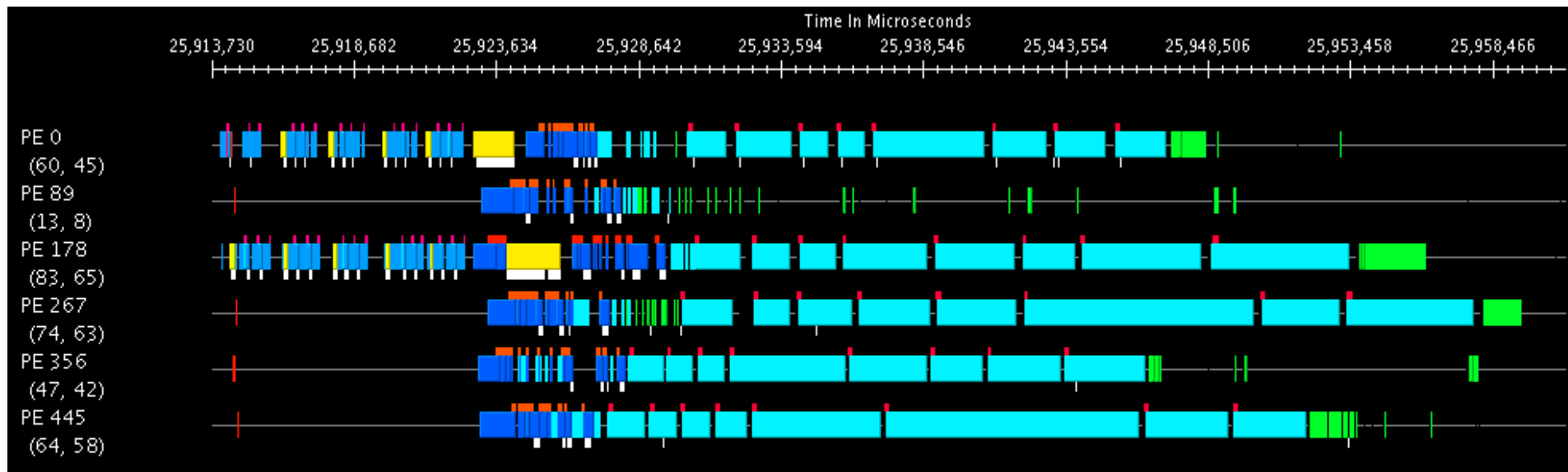
Timeline



Most common view. Much more detailed than overview.

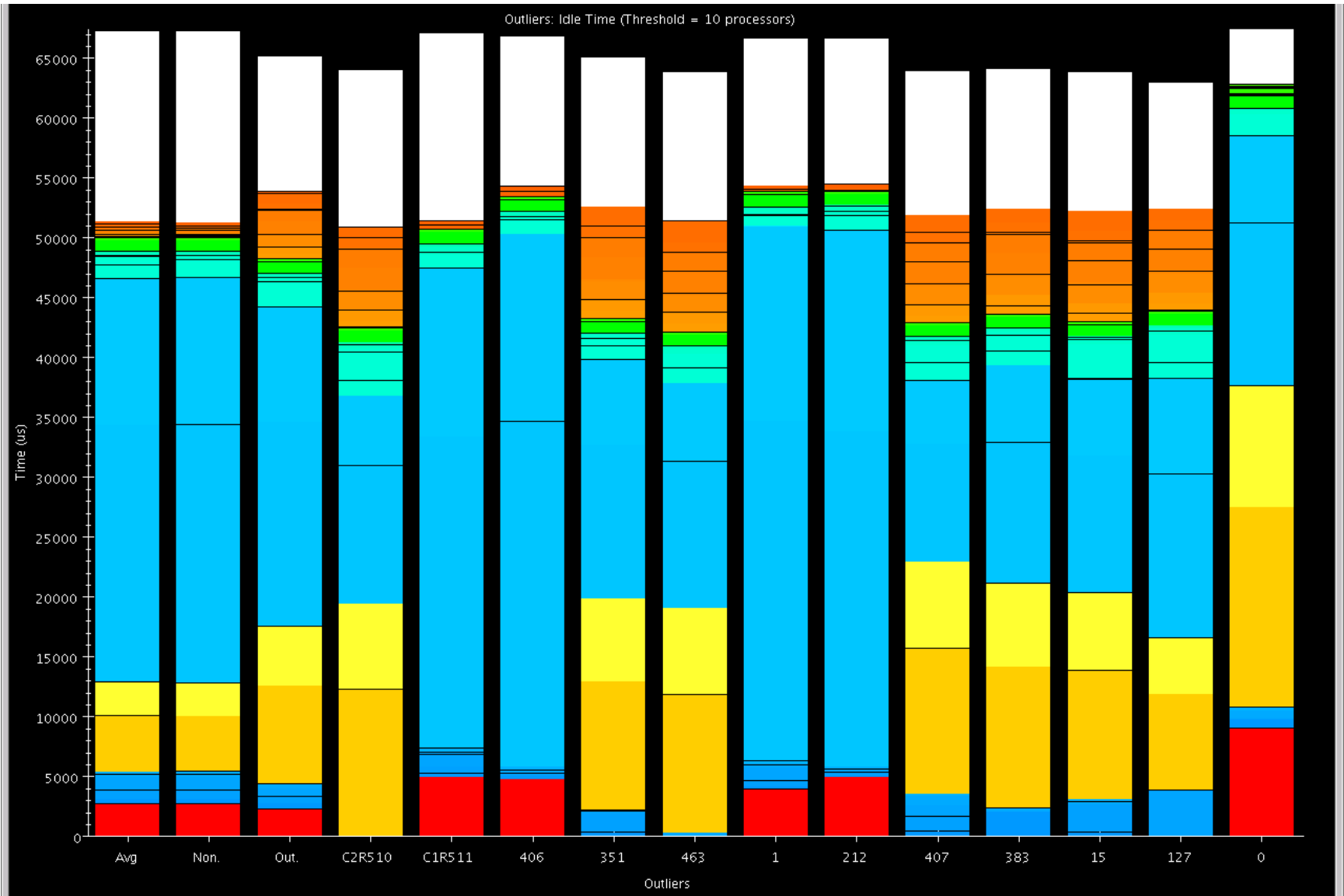


Clicking on EPs traces messages,
mouseover shows EP details.

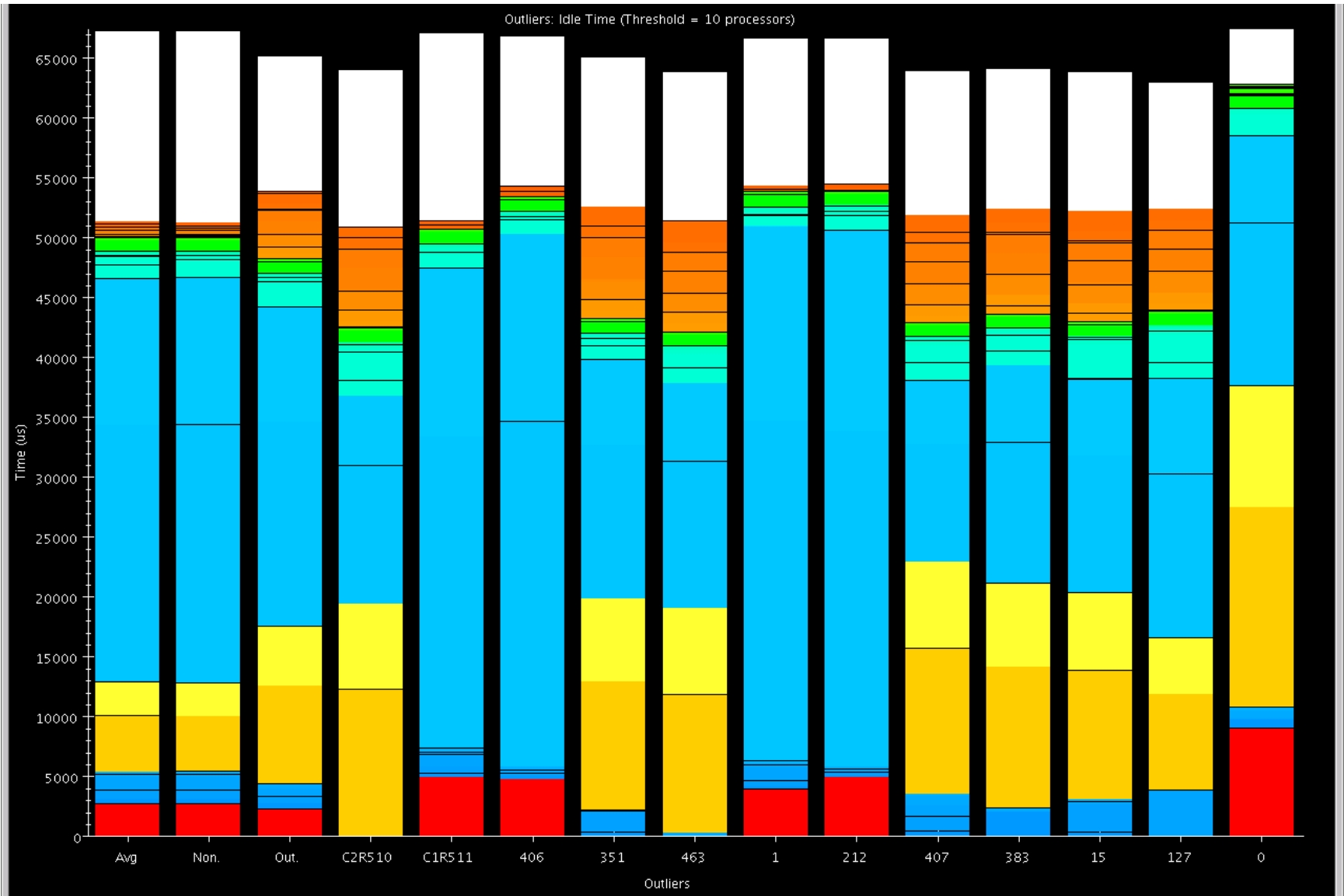


Colors are different EPs. White ticks on bottom represent message sends, red ticks on top represent user events.

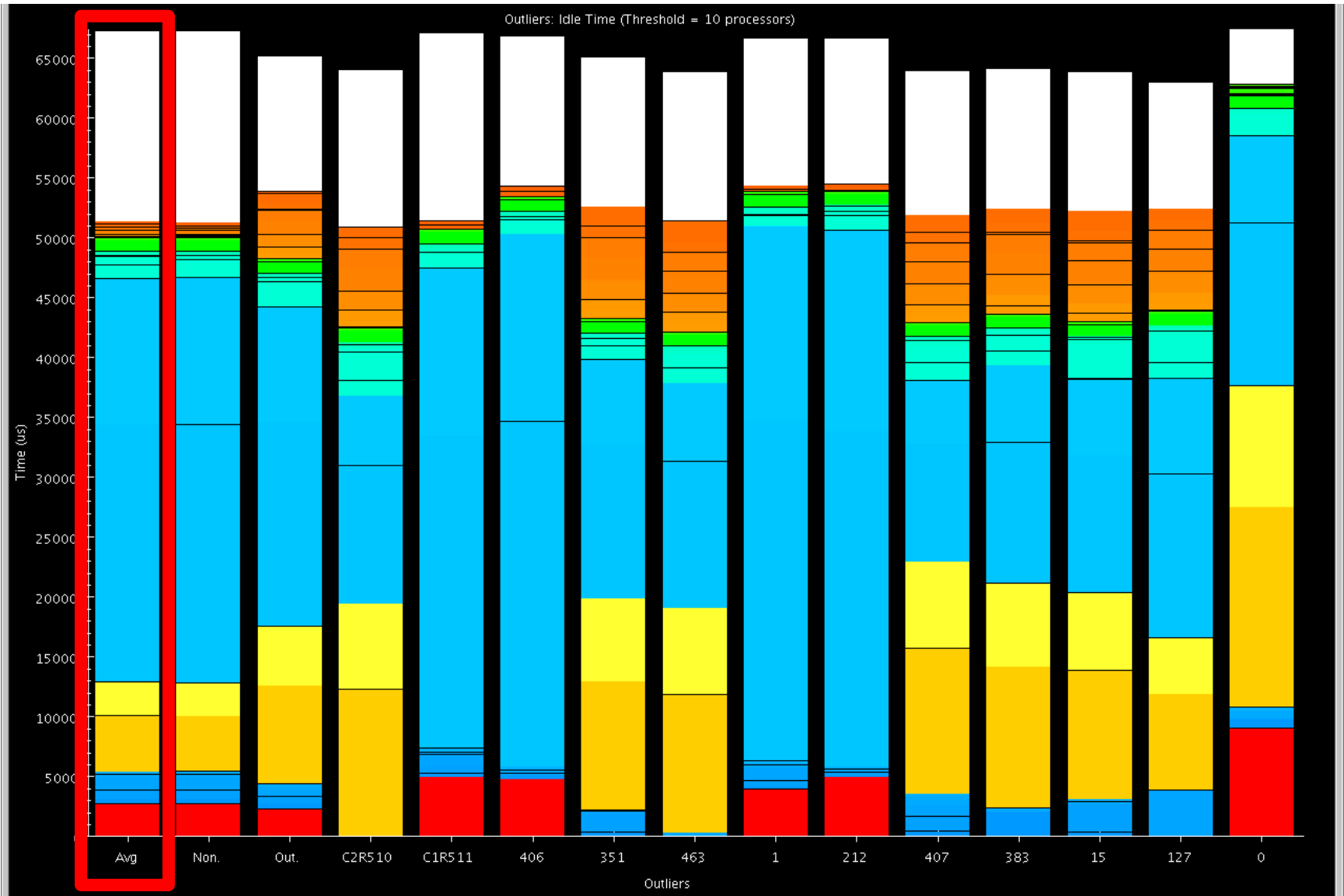
Processed Data Views



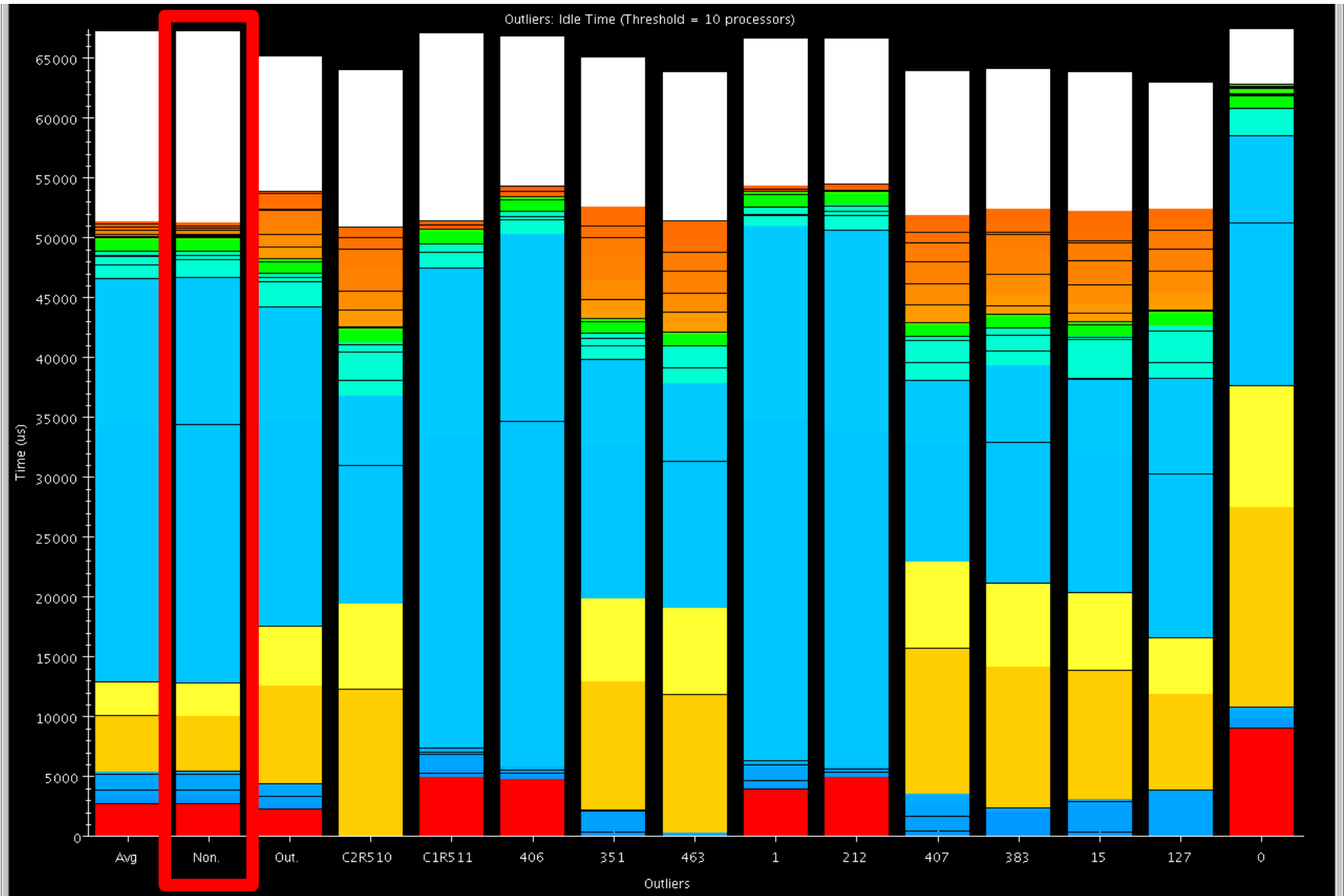
Outlier Analysis



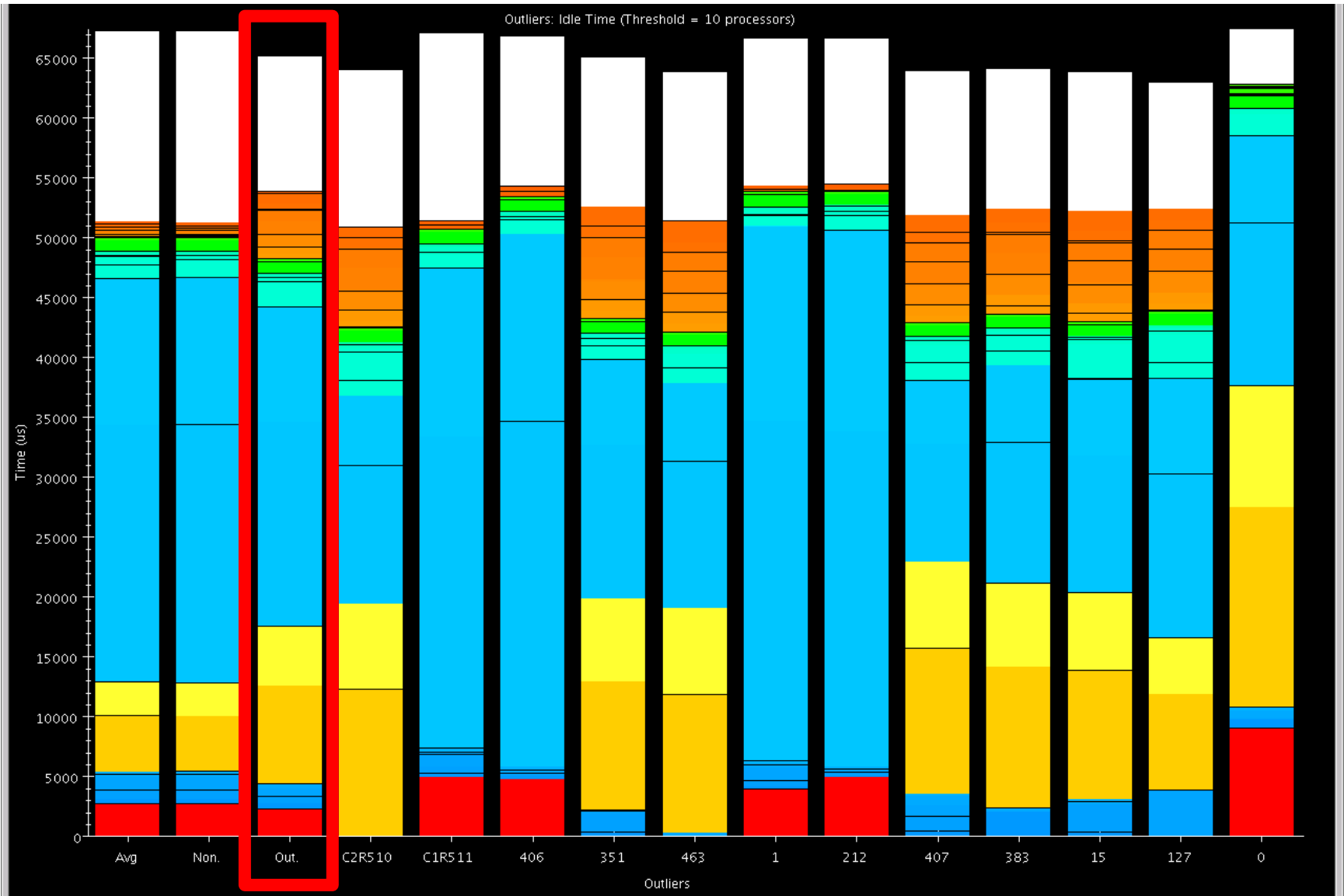
k-Means to find “extreme” processors



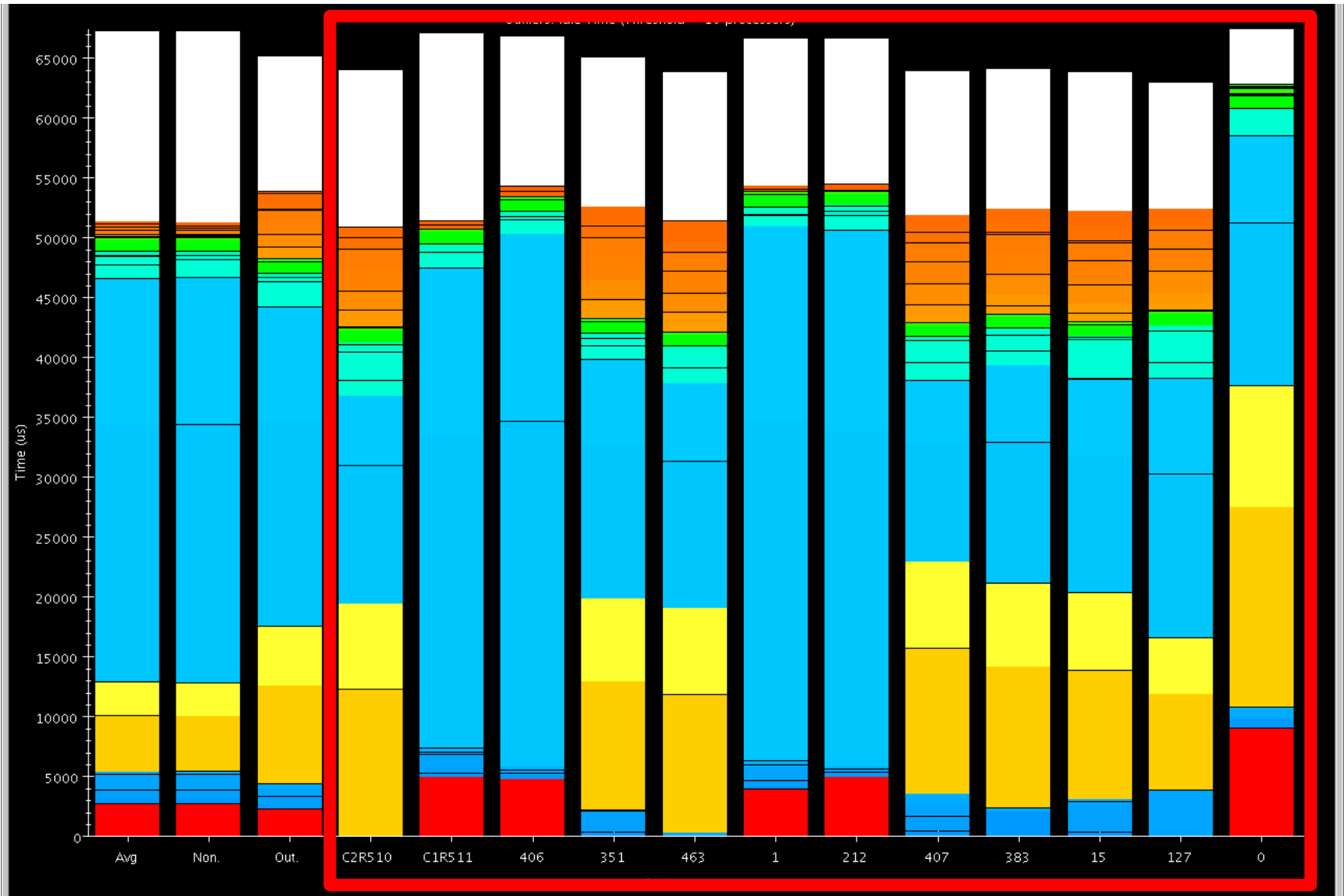
Global Average



Non-Outlier Average



Outlier Average



Cluster Representatives and Outliers

Advanced Features

- Live Streaming
 - Run server from job to send performance traces in real time
- Online Extrema Analysis
 - Perform clustering during job; only save representatives and outliers
- Multirun Analysis
 - Side by side comparison of data from multiple runs

Future Directions

- PICS - expose application settings to RTS for on the fly tuning
- End of run analysis - use remaining time after job completion to process performance logs
- Simulation - Increased reliance on simulation for generating performance logs

Conclusions

- Projections has been used to effectively solve performance woes
- Constantly improving the tools
- Scalable analysis is become increasingly important