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 – User Stats

In addition to user events, users can add events with custom values as User Stats.

Register Stat:
int traceRegisterUserStat(const char* EventDesc, int StatNum)
Update Stat:
void updateStat(int StatNum, double StatValue)
Update a Stat Pair:
void updateStatPair(int EventNum, double StatValue, double Time)

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

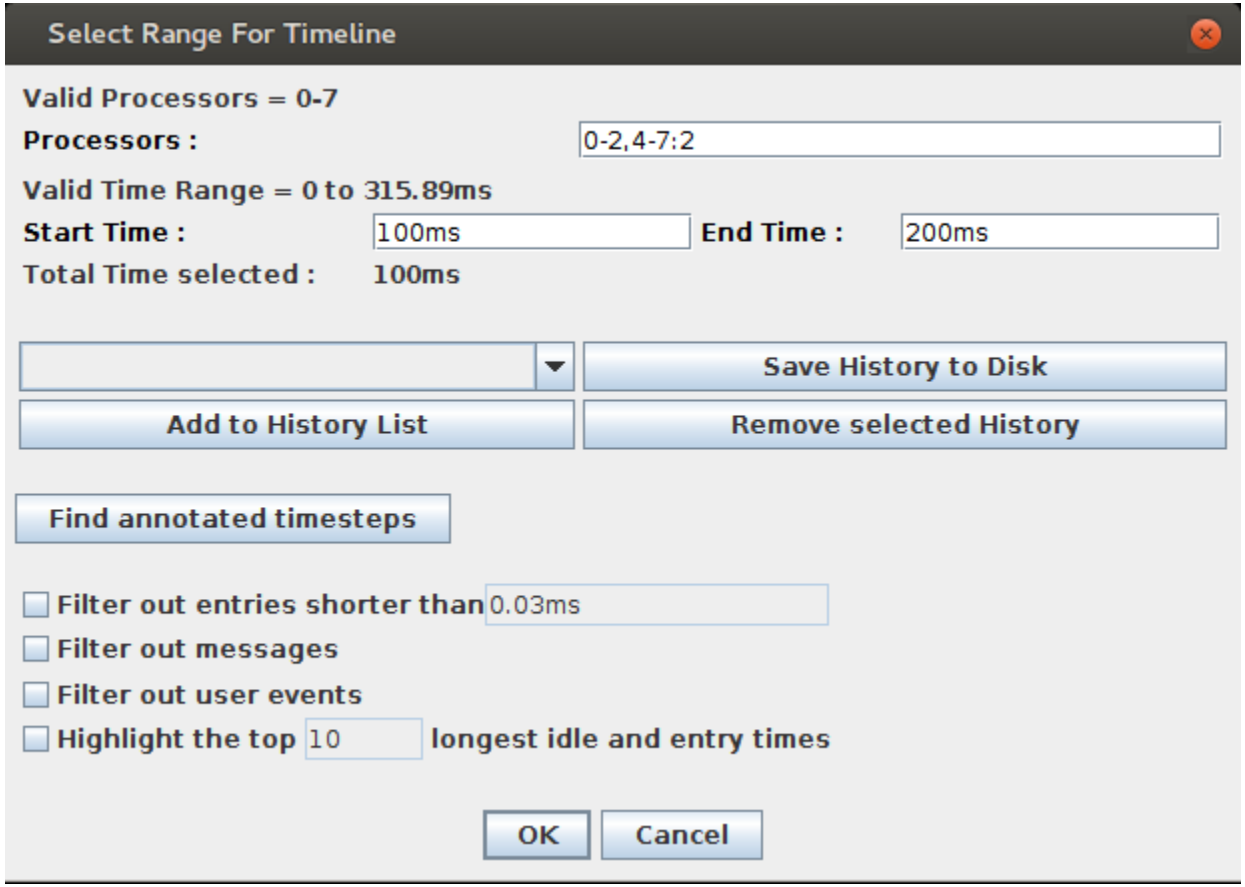
- Tools of aggregated performance viewing
  - Time profile
  - Histogram
  - Communication
- Tools of processor level granularity
  - Overview
  - Timeline
- Tools of derived/processed data
  - Outlier analysis: identifies outliers

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.

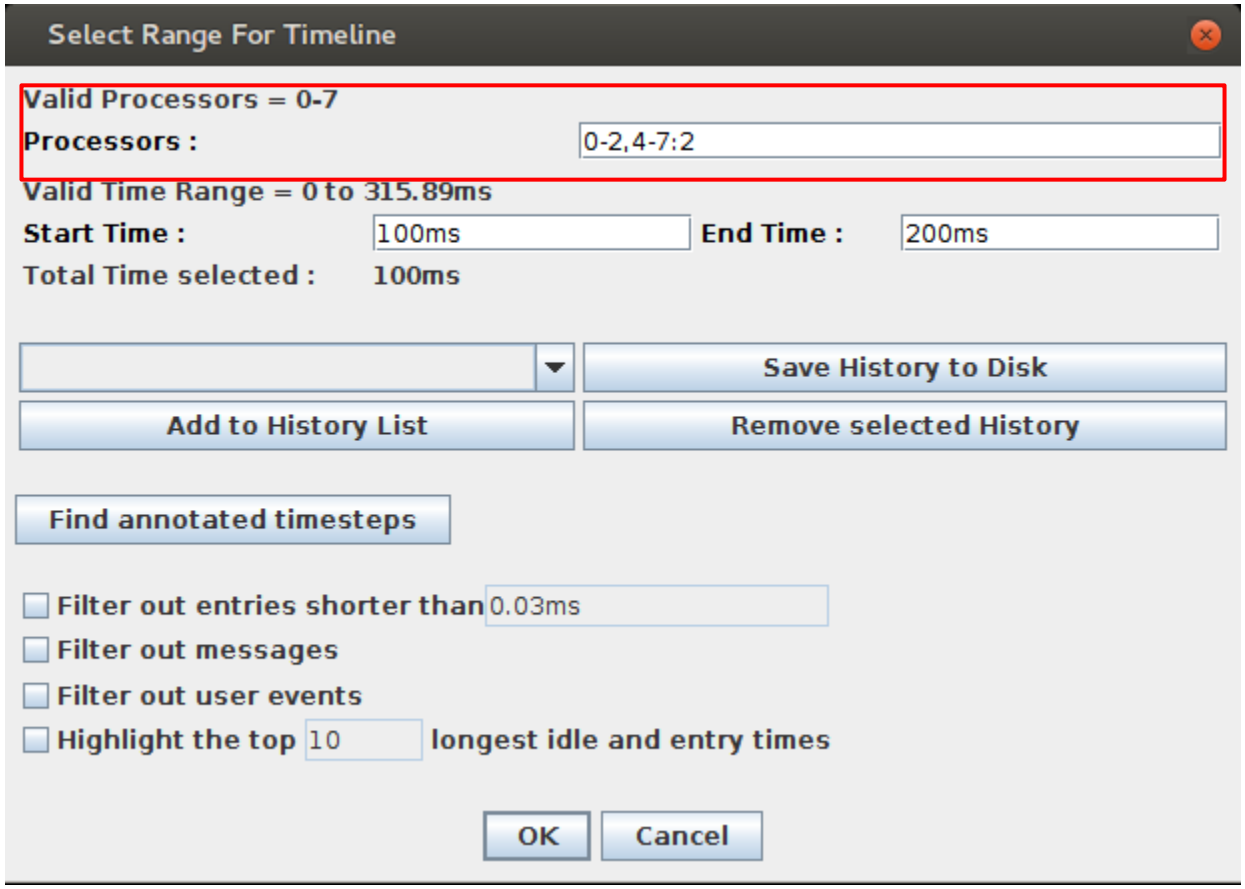
Analysis Techniques

- Zoom in/out to find potential problem spots.
- Mouseover graohs 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 :
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

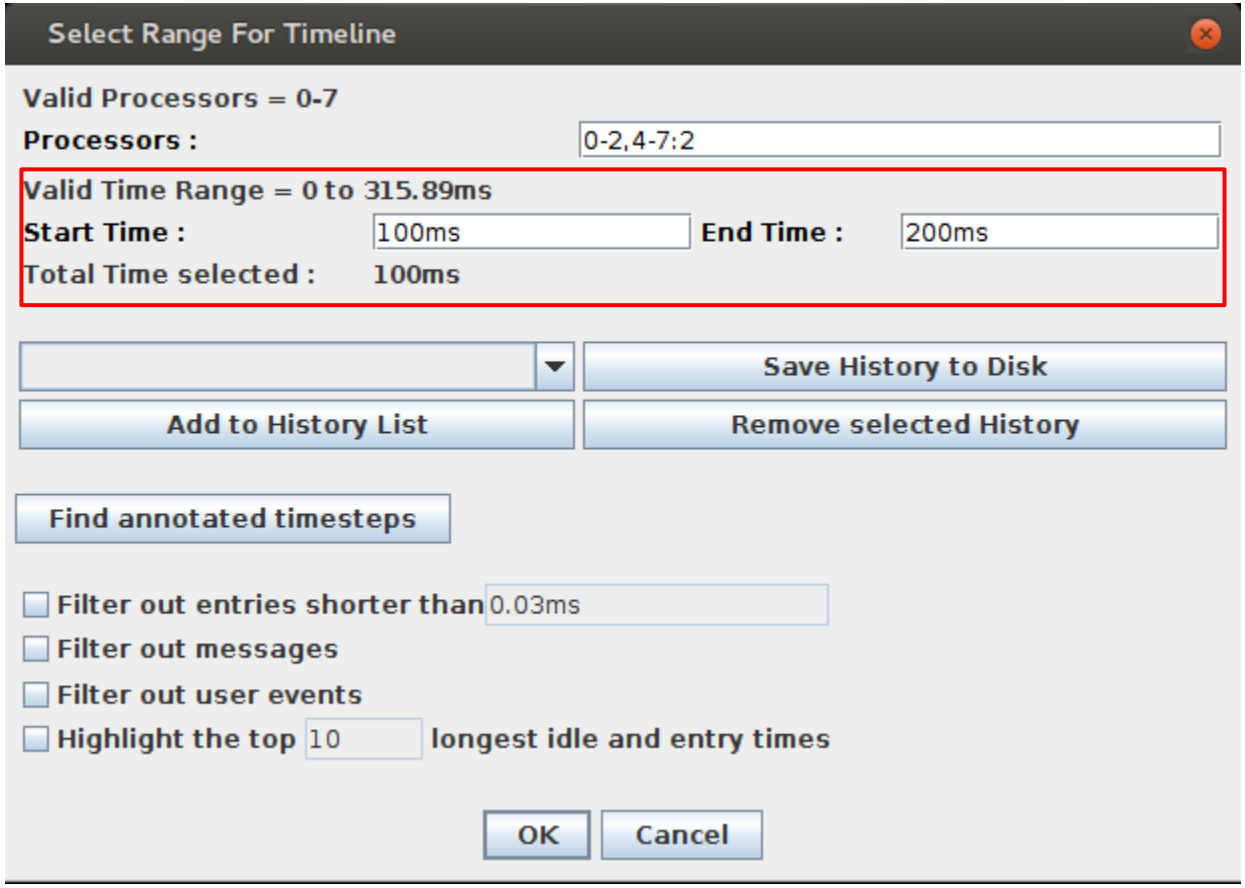
Dialog Box

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



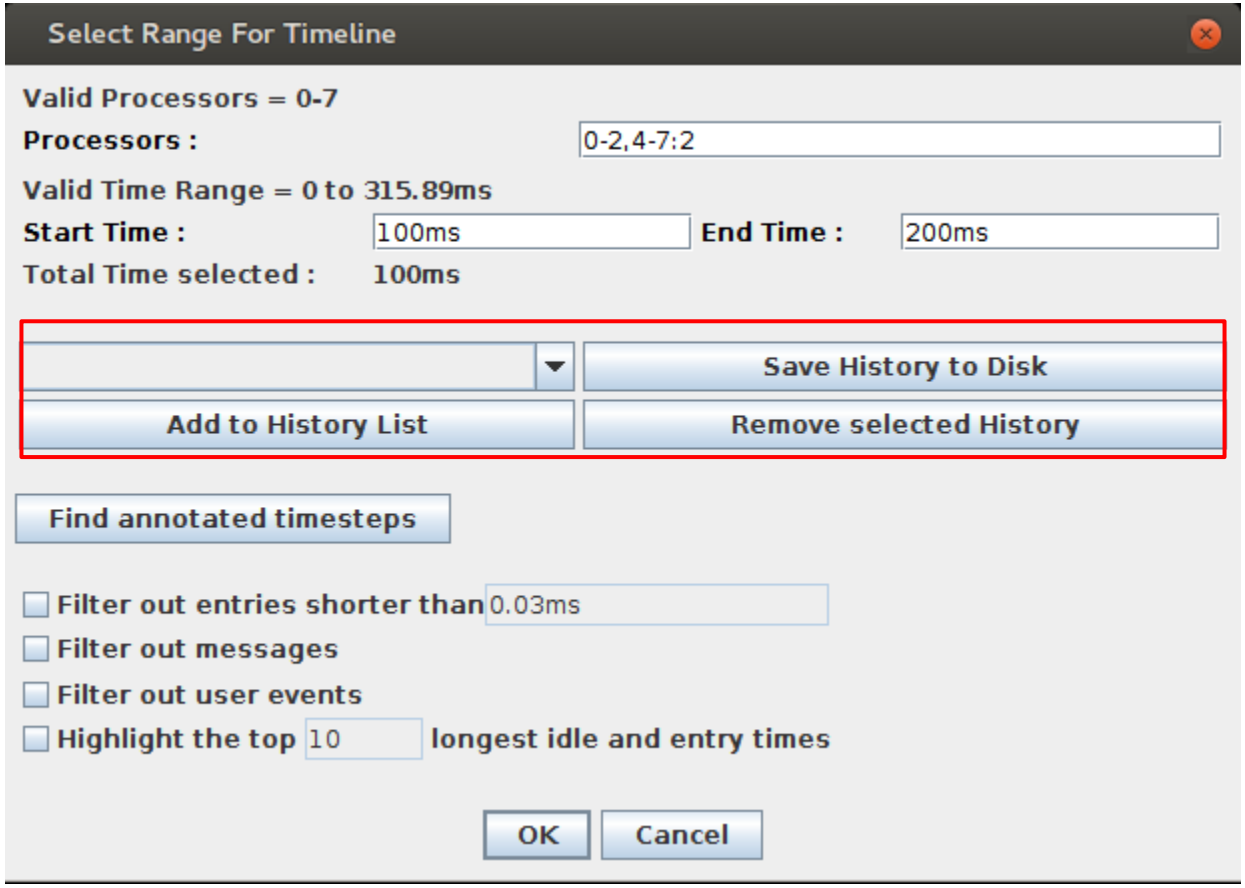
Dialog Box

Select time range



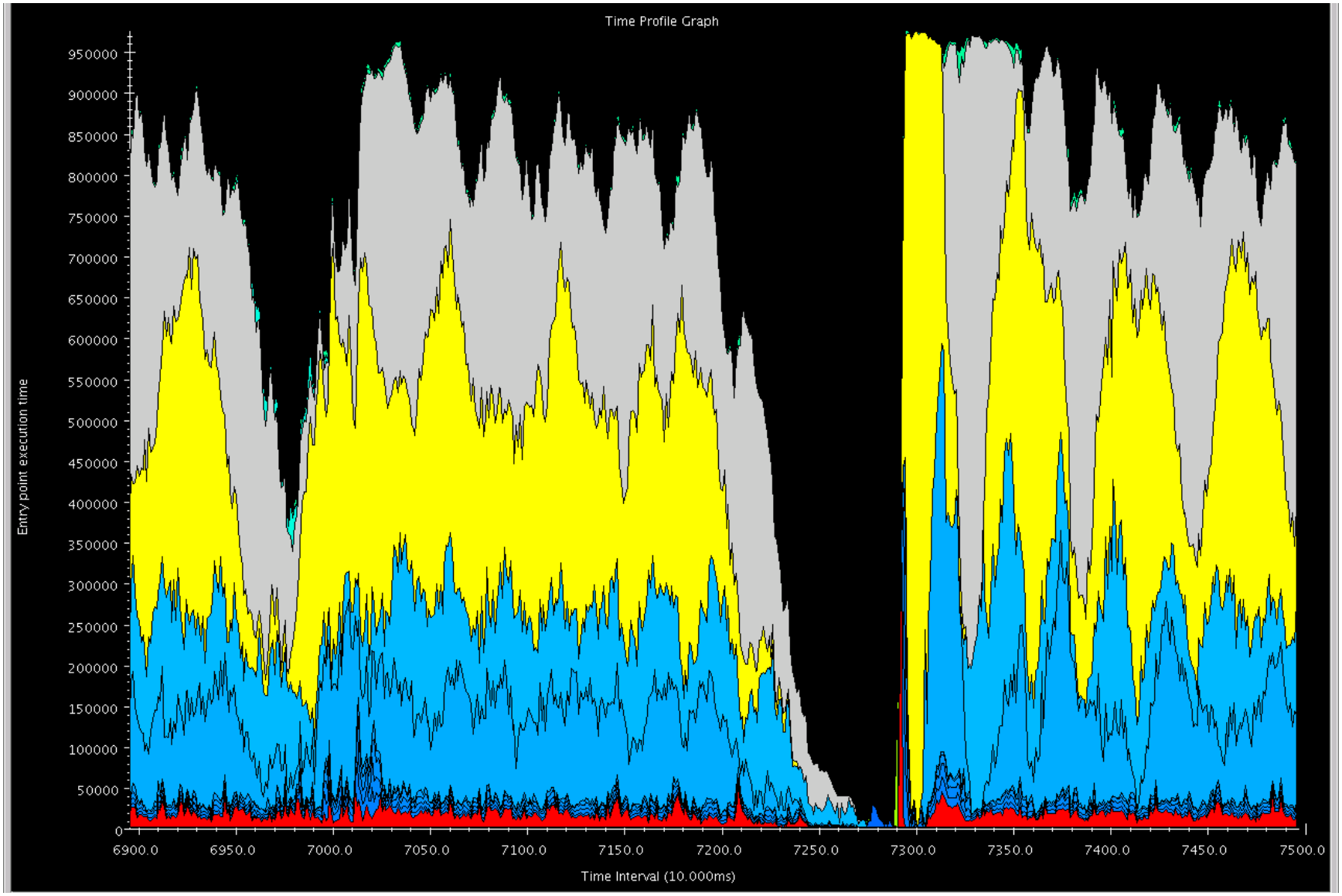
Dialog Box

Add presets to history

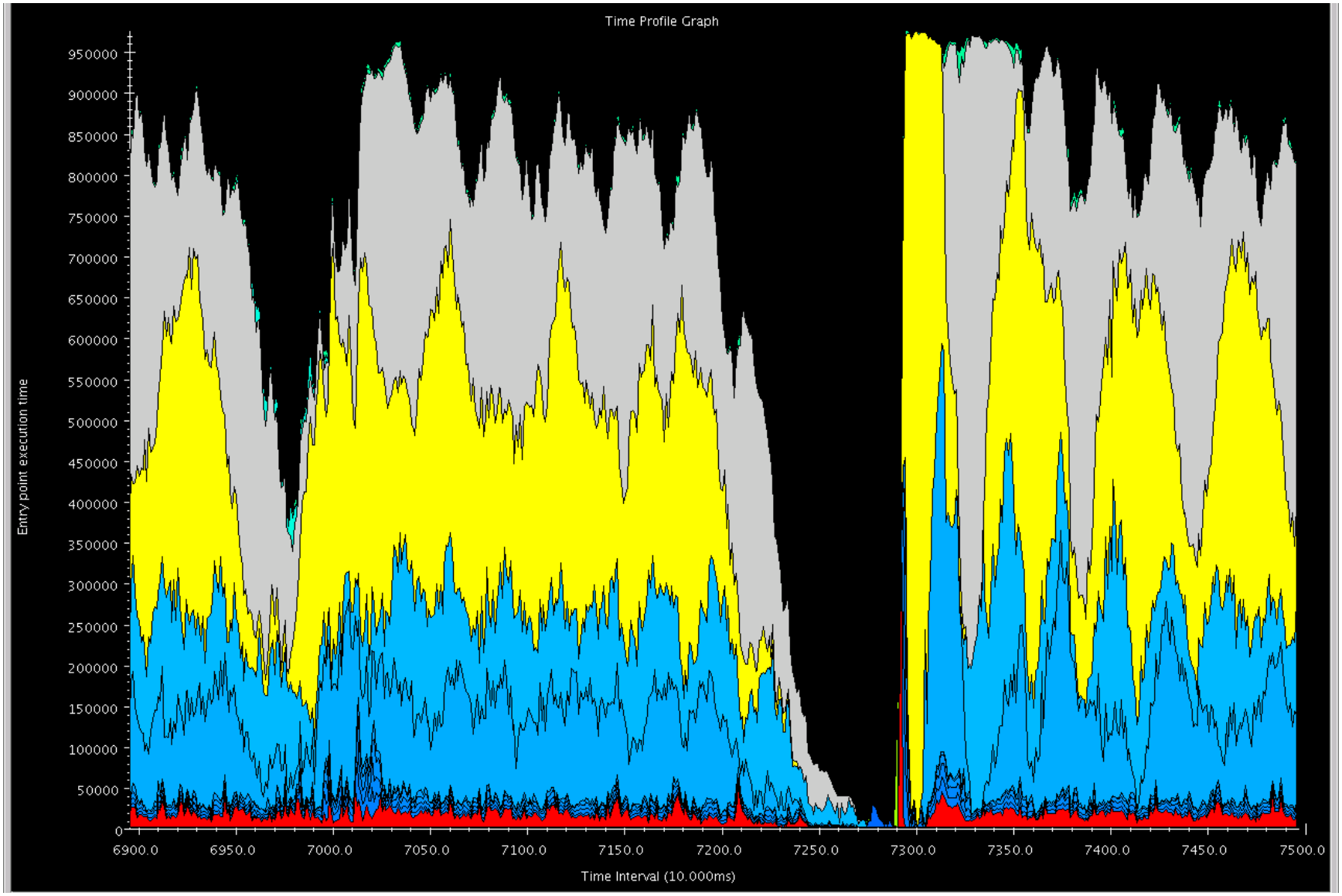


Dialog Box

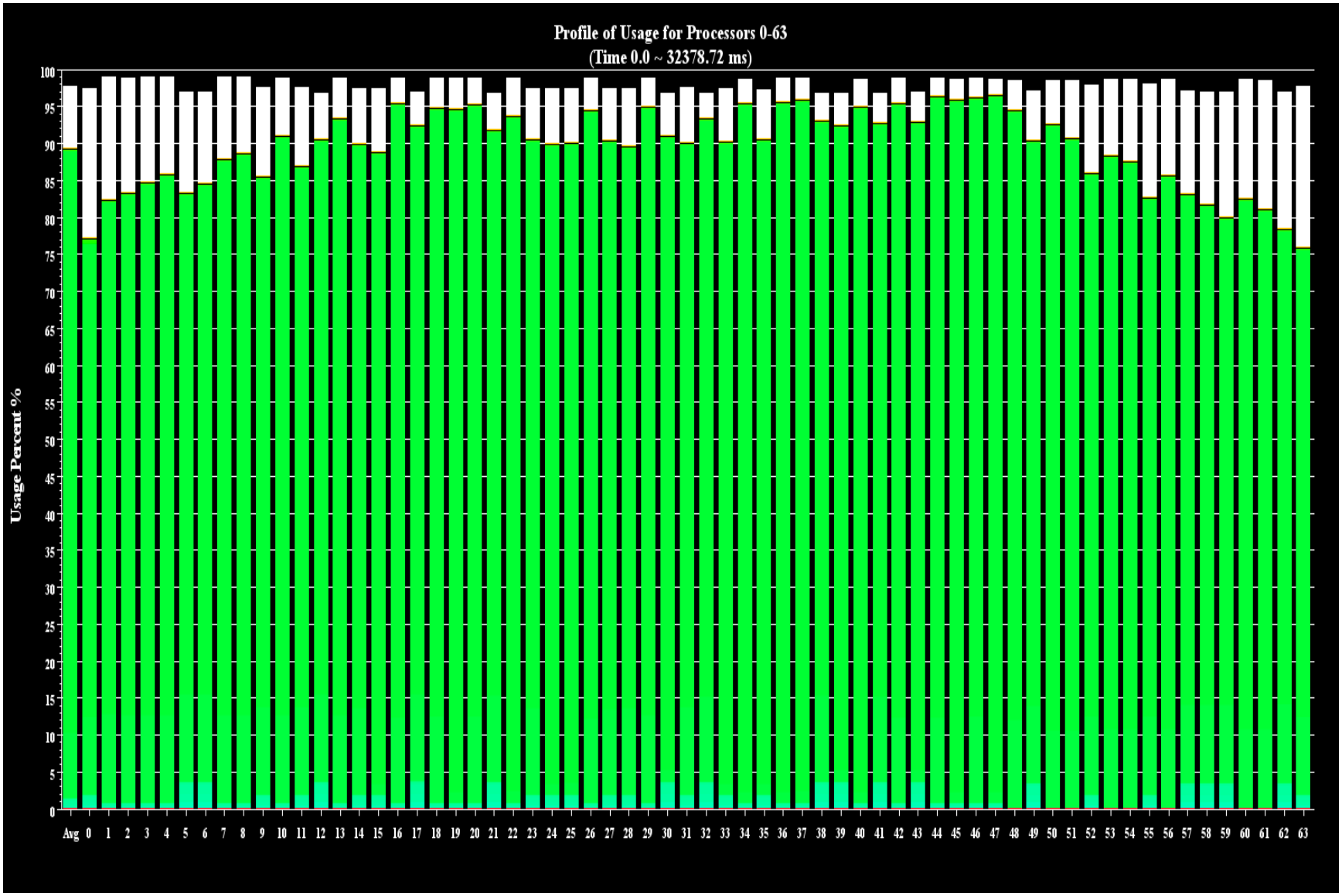
Aggregate Views



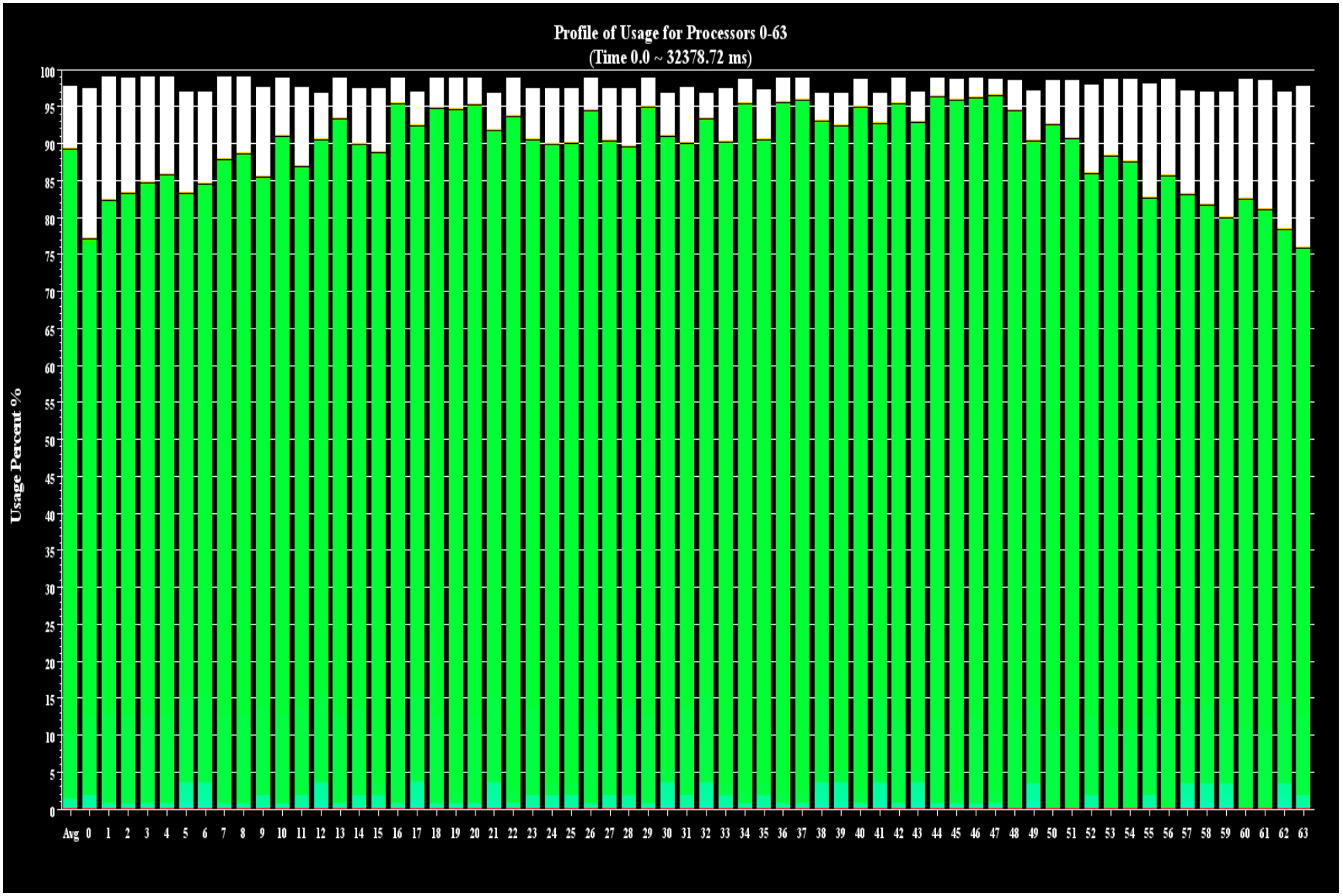
Time Profile



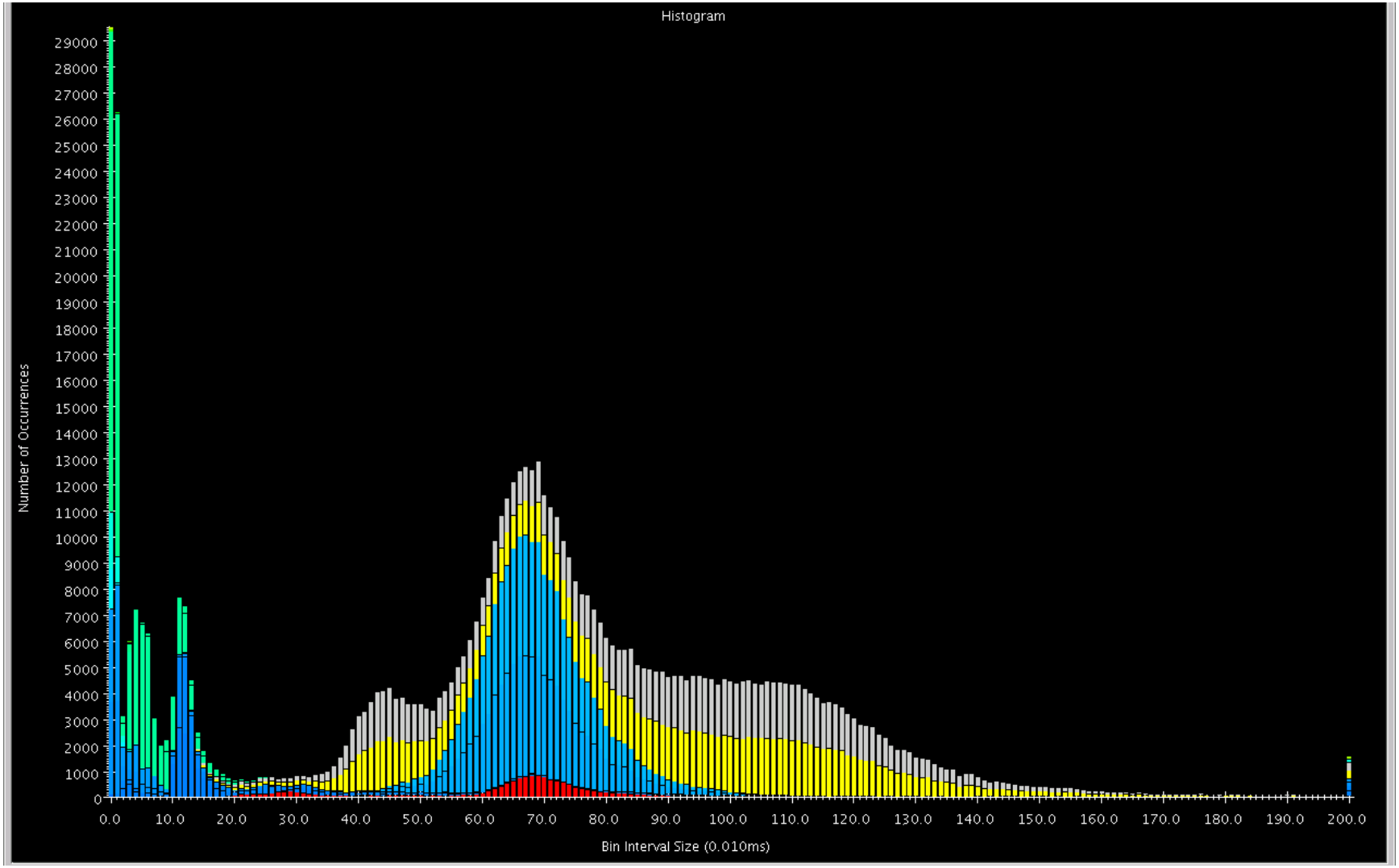
Time spent by each EP summed across all PEs in time interval



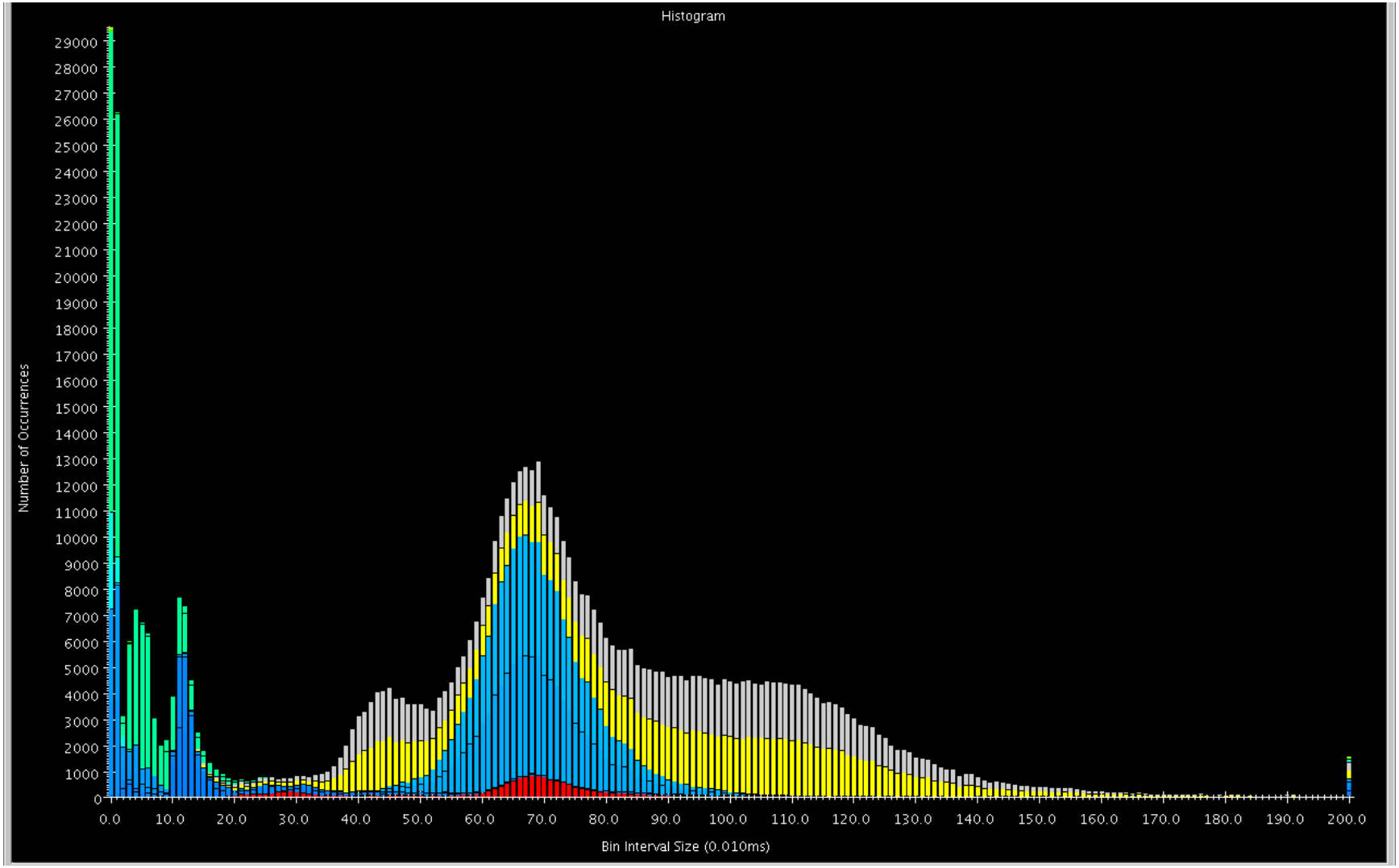
Usage Profile



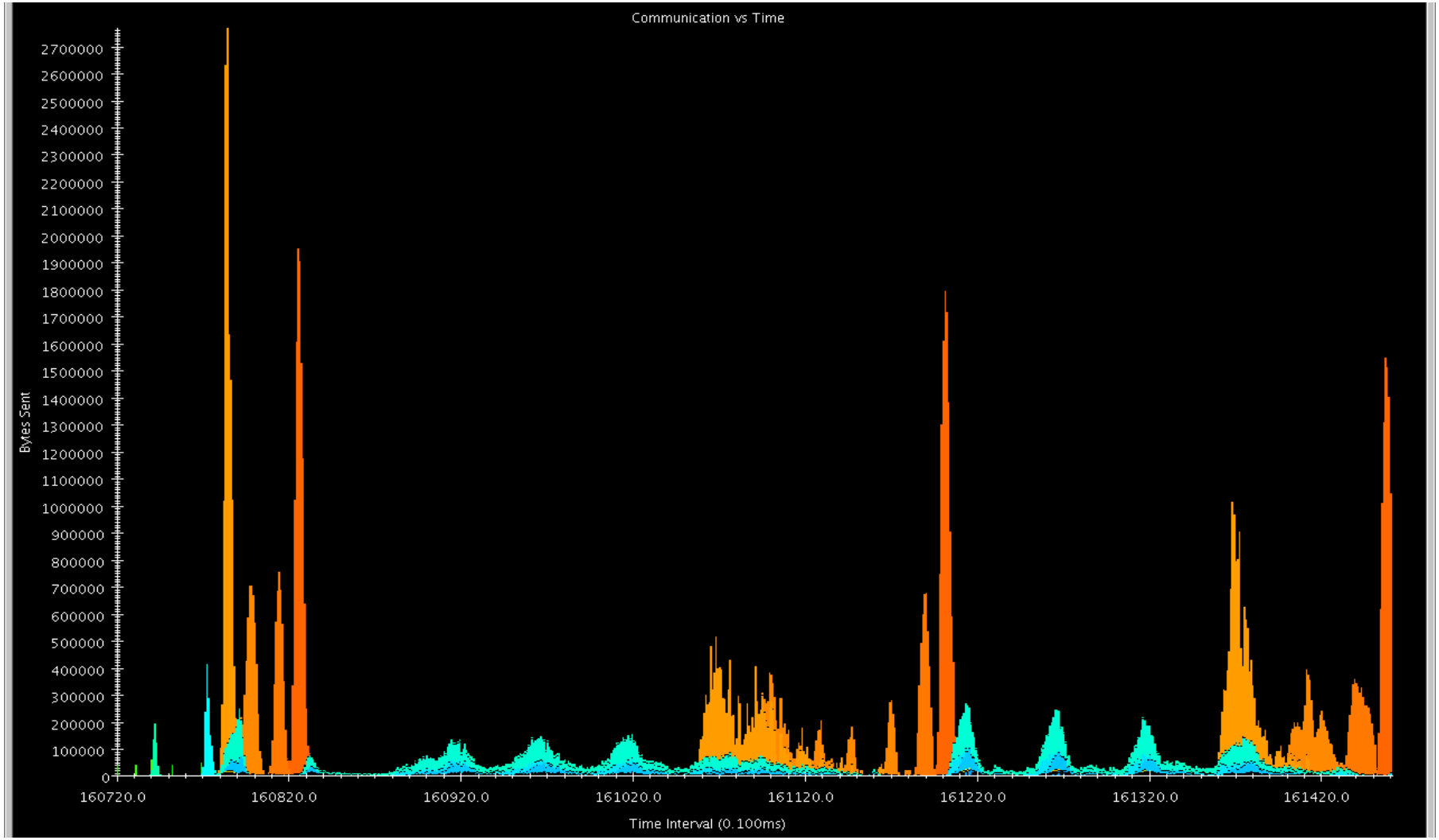
Percent utilization per PE over interval


Histogram
Number of Occurrences
29000
28000
27000
26000
25000
24000
23000
22000
21000
20000
19000
18000
17000
16000
15000
14000
13000
12000
11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
110.0
120.0
130.0
140.0
150.0
160.0
170.0
180.0
190.0
200.0
Bin Interval Size (0.010ms)

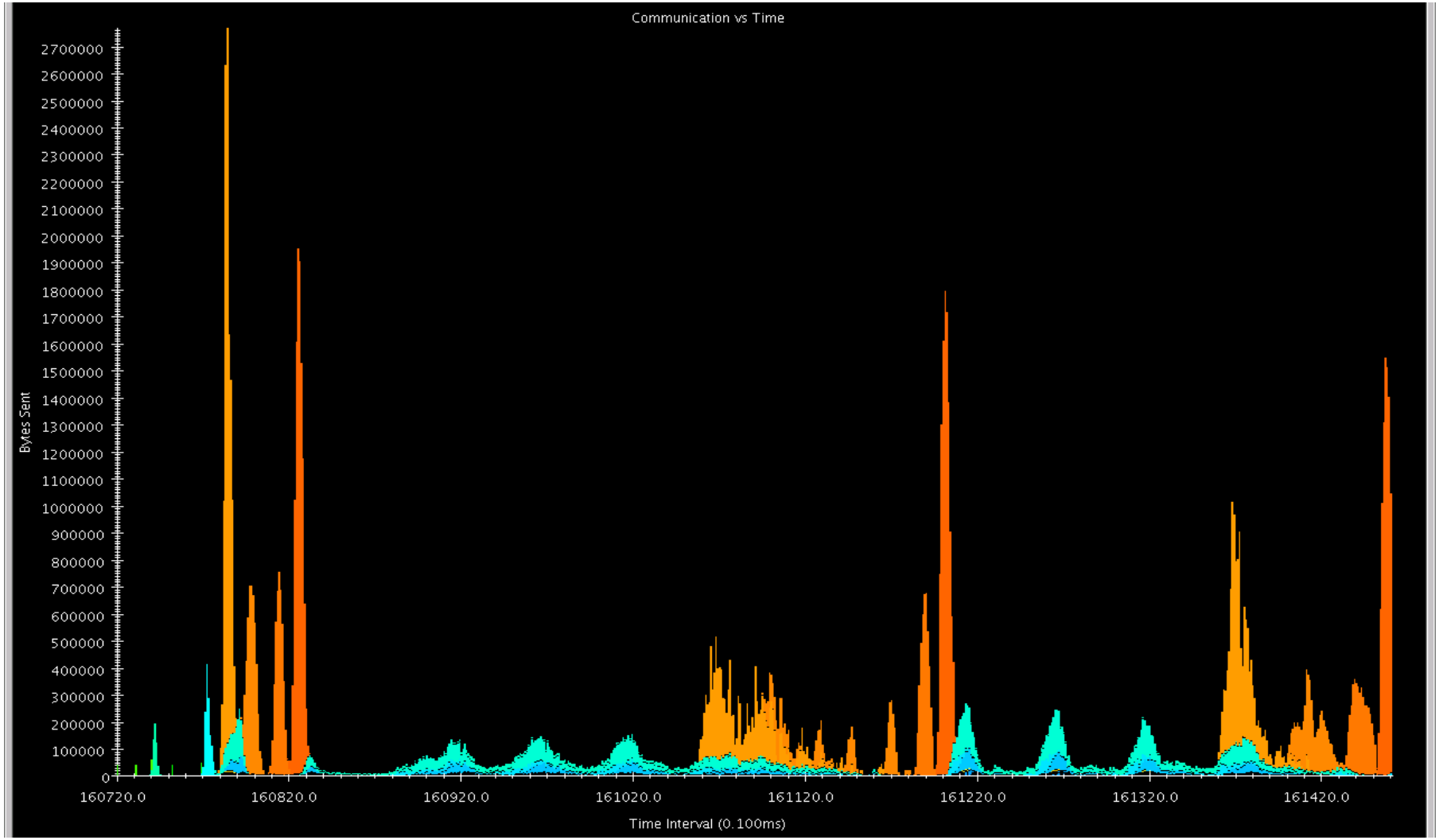
Histogram



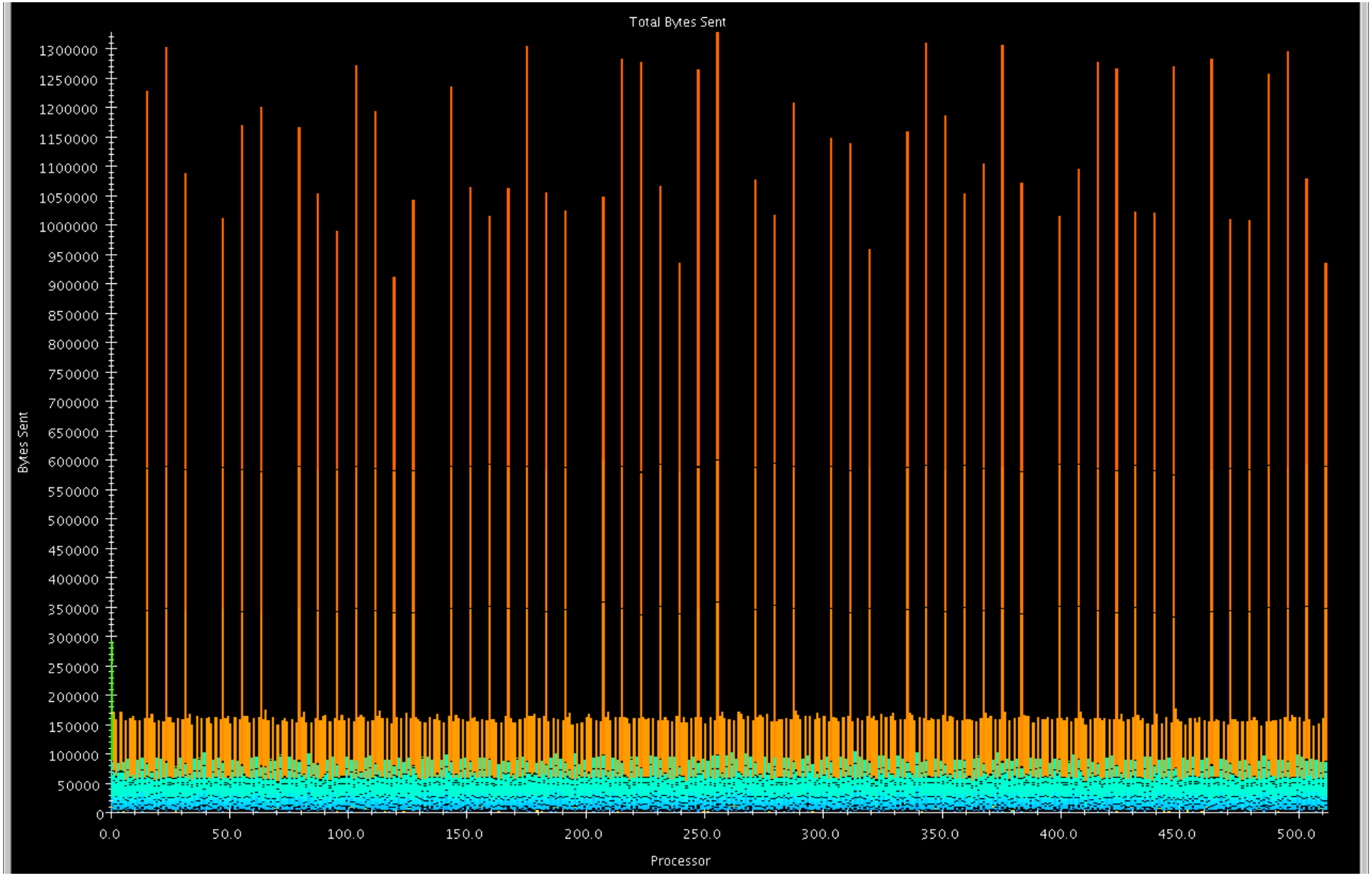
Shows statistics in “frequency” domain.



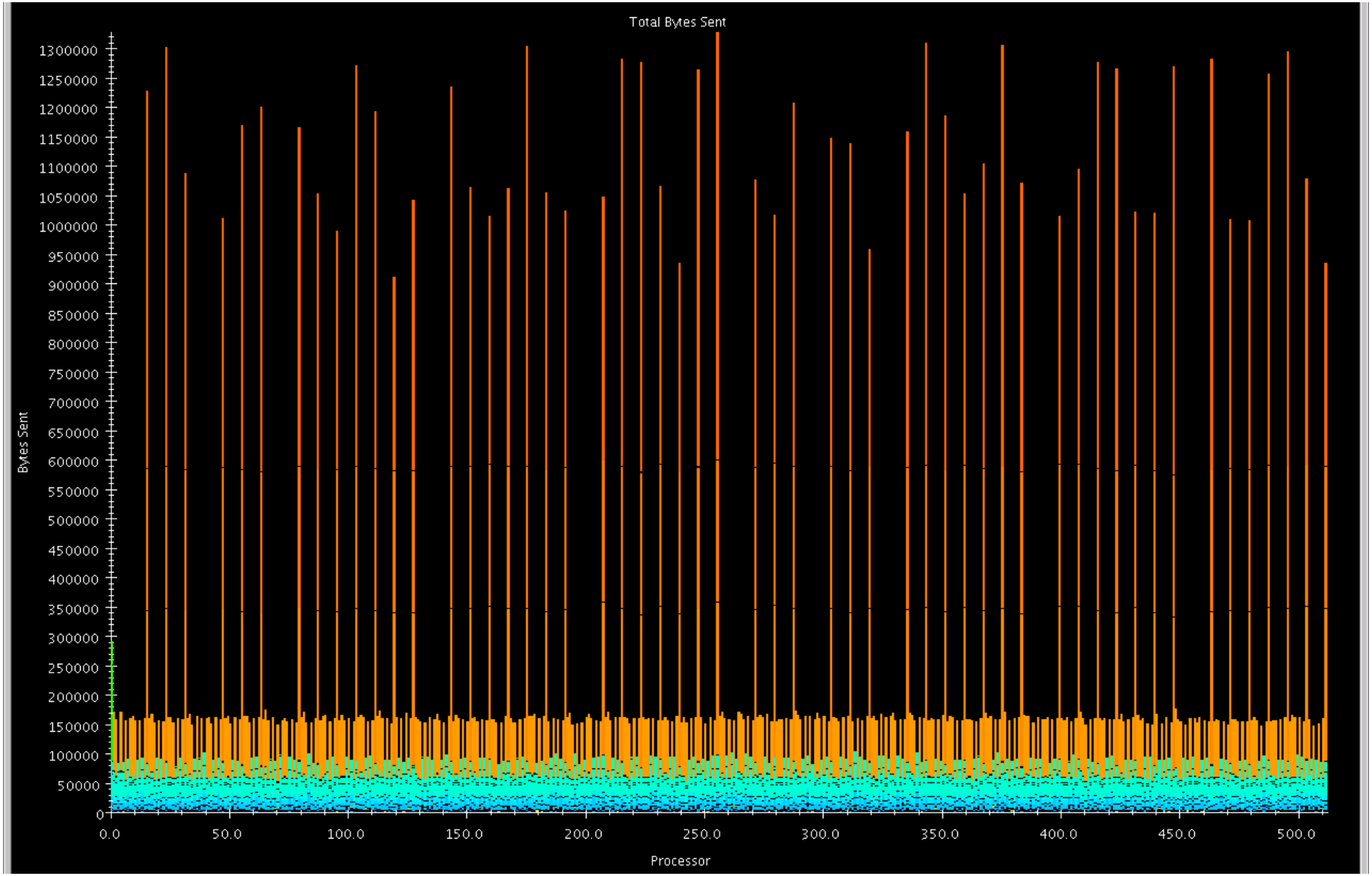
Communication vs. Time



Shows communication over all PEs in the time domain.

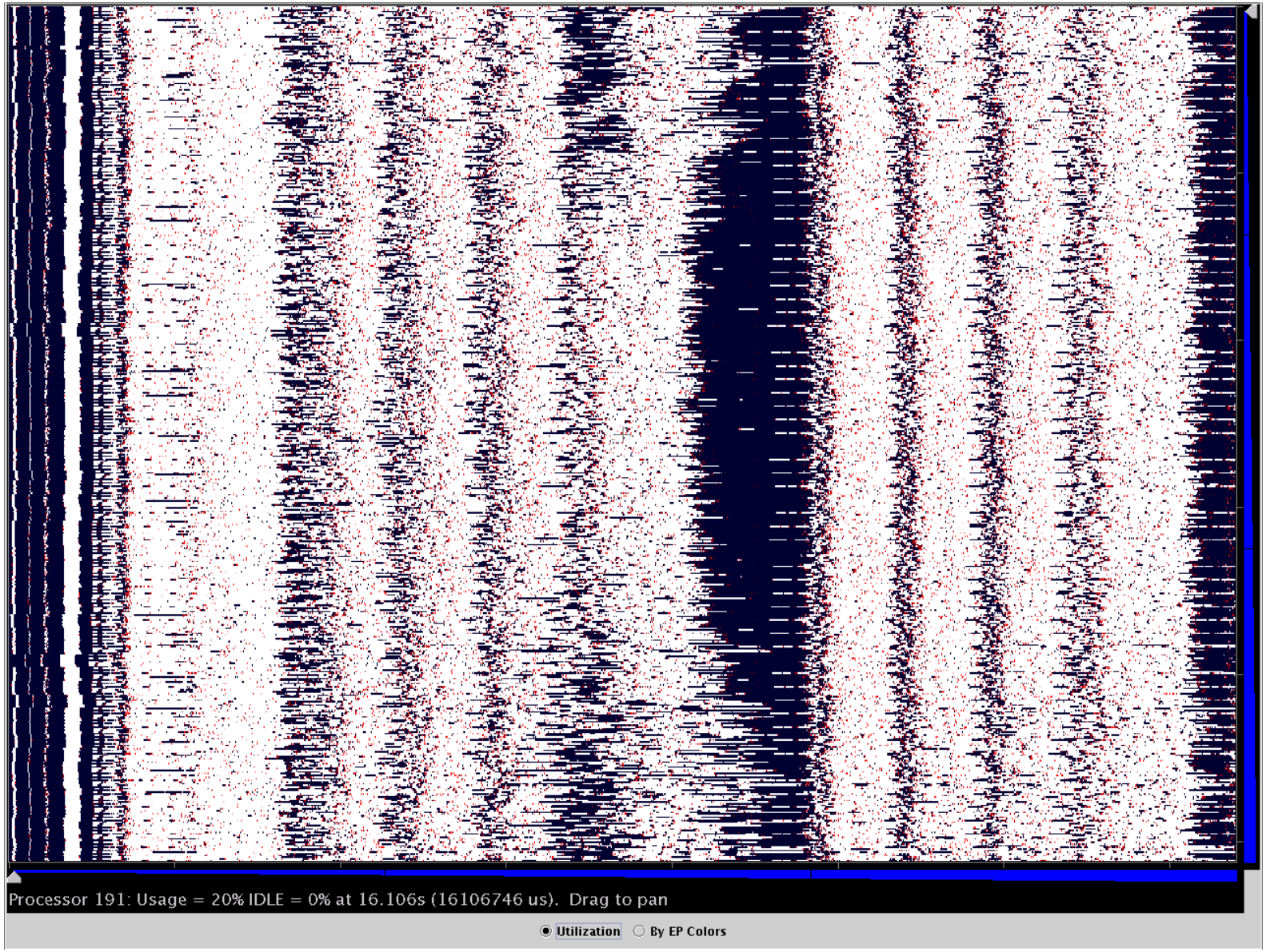


Communication per Processor

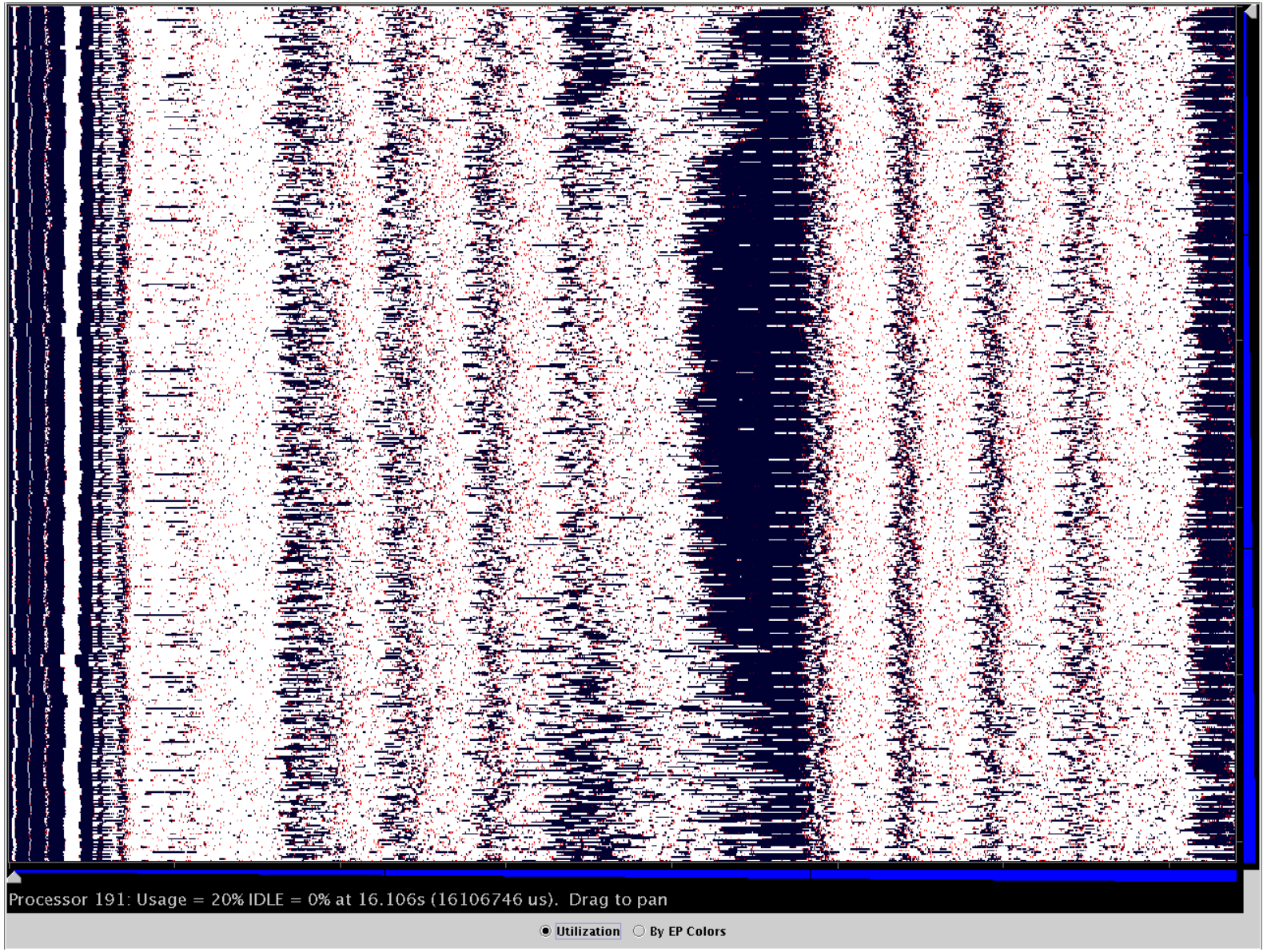


Shows how much each PE communicated over the whole job.

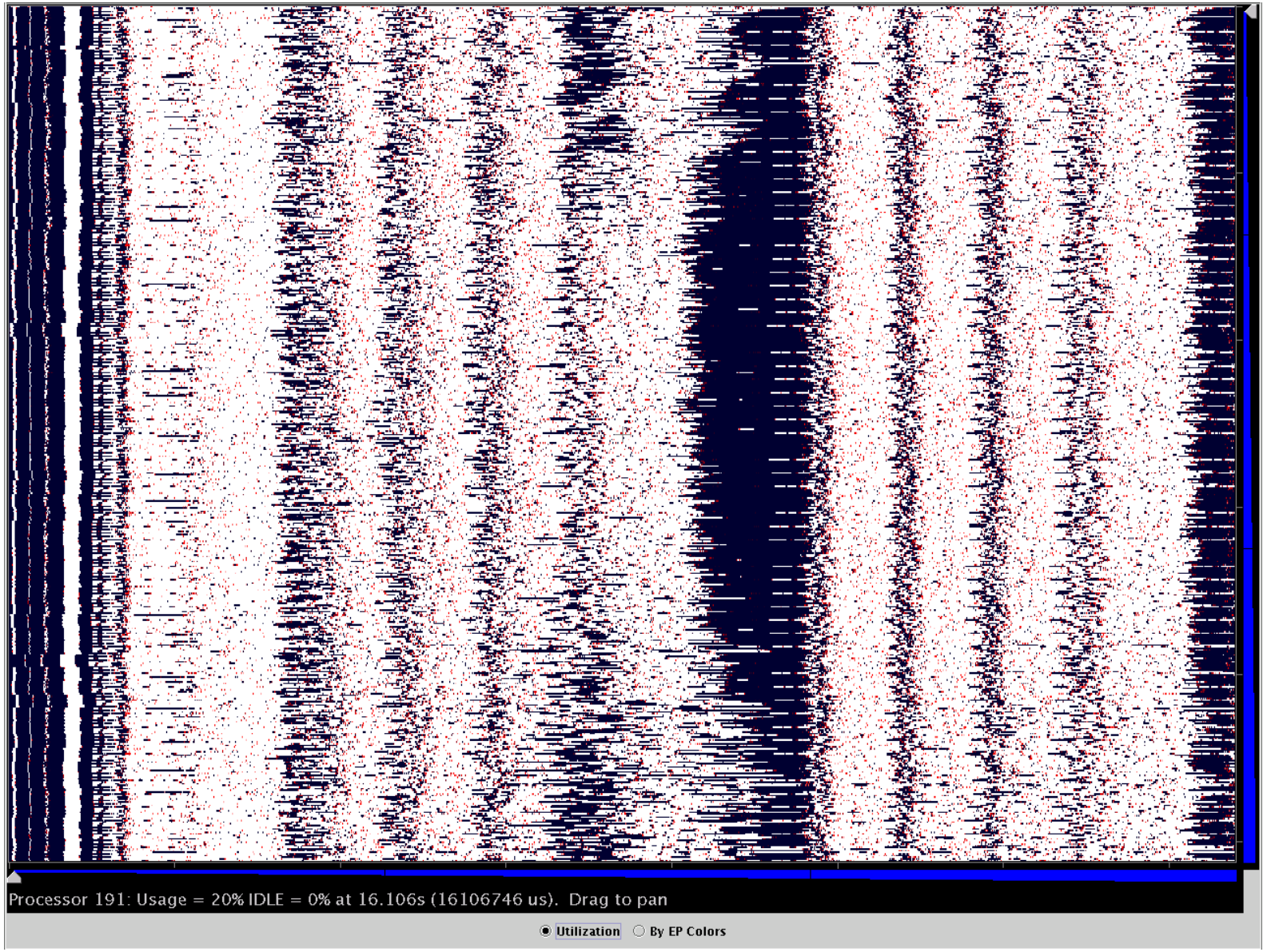
Processor Level Views


Processor 191: Usage = 20% IDLE = 0% at 16.106s (16106746 us). Drag to pan
Utilization
By EP Colors

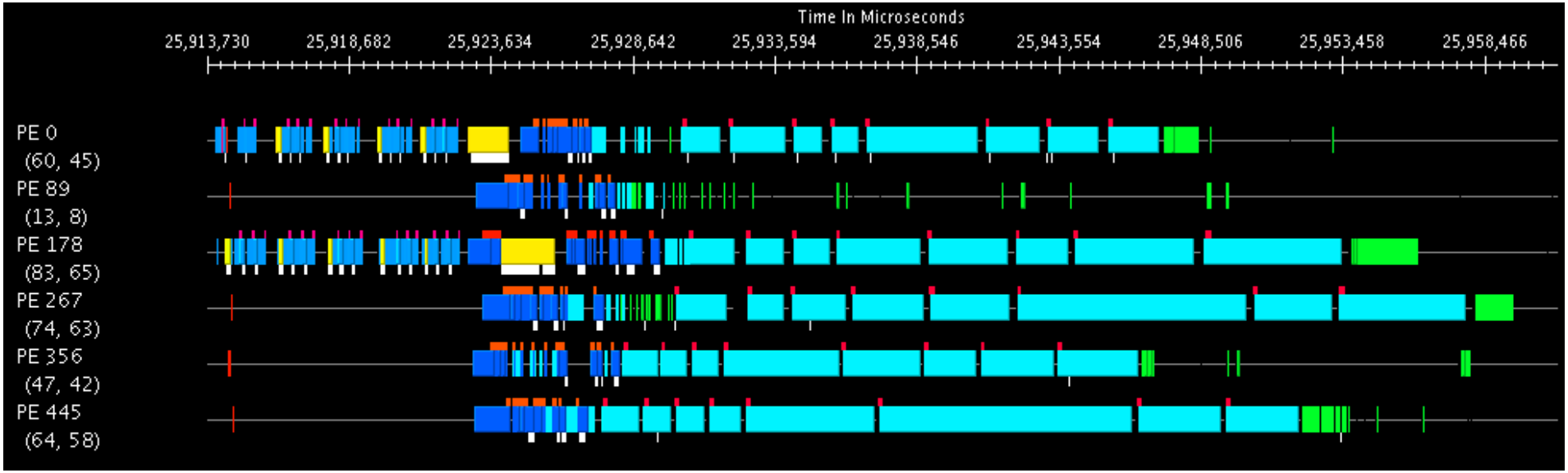
Overview



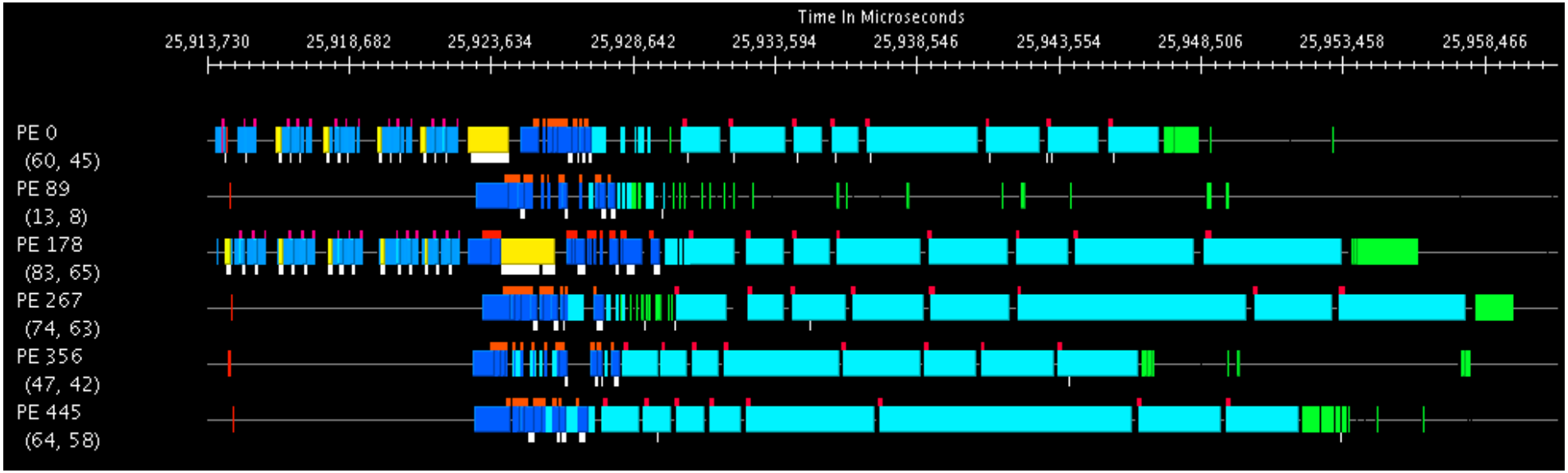
Time on X, different PEs on Y



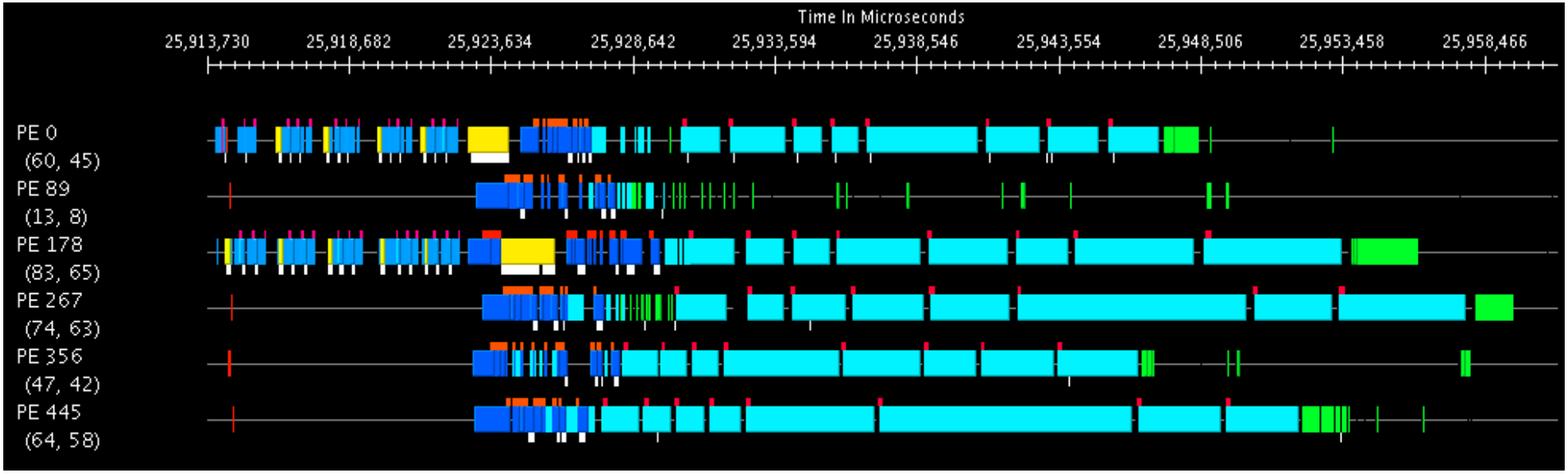
Intensity of plot represents PE's utilization at that time


Time In Microseconds
25,913,730
25,918,682
25,923,634
25,928,642
25,933,594
25,938,546
25,943,554
25,948,506
25,953,458
25,958,466
PE 0
(60, 45)
PE 89
(13, 8)
PE 178
(83, 65)
PE 267
(74, 63)
PE 356
(47, 42)
PE 445
(64, 58)

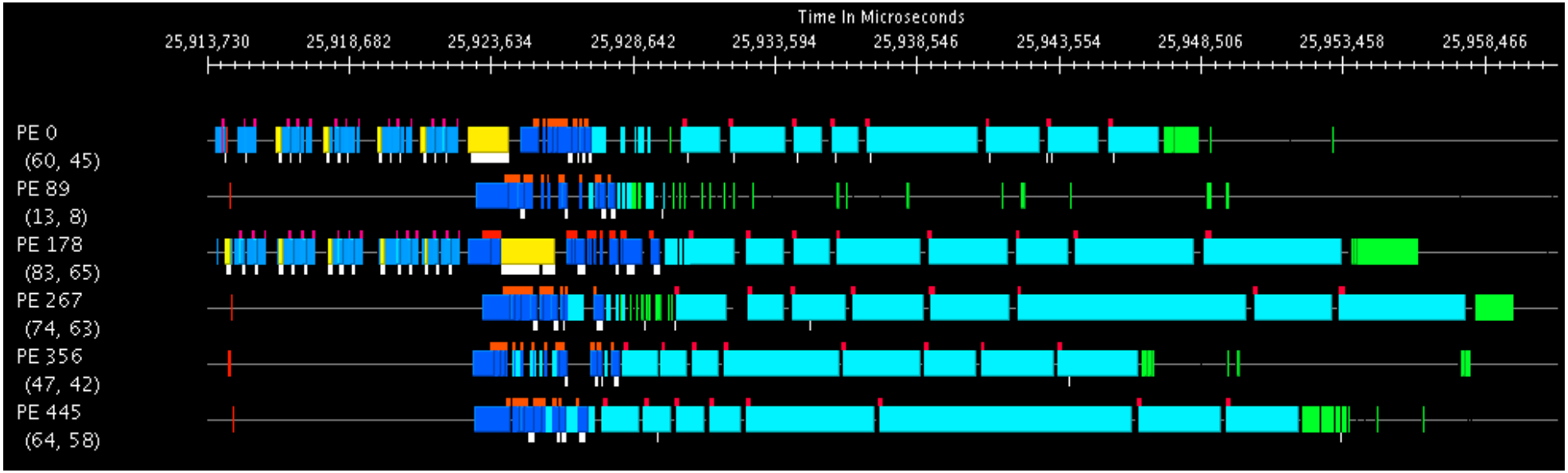
Timeline



Most common view. Much more detailed than overview.

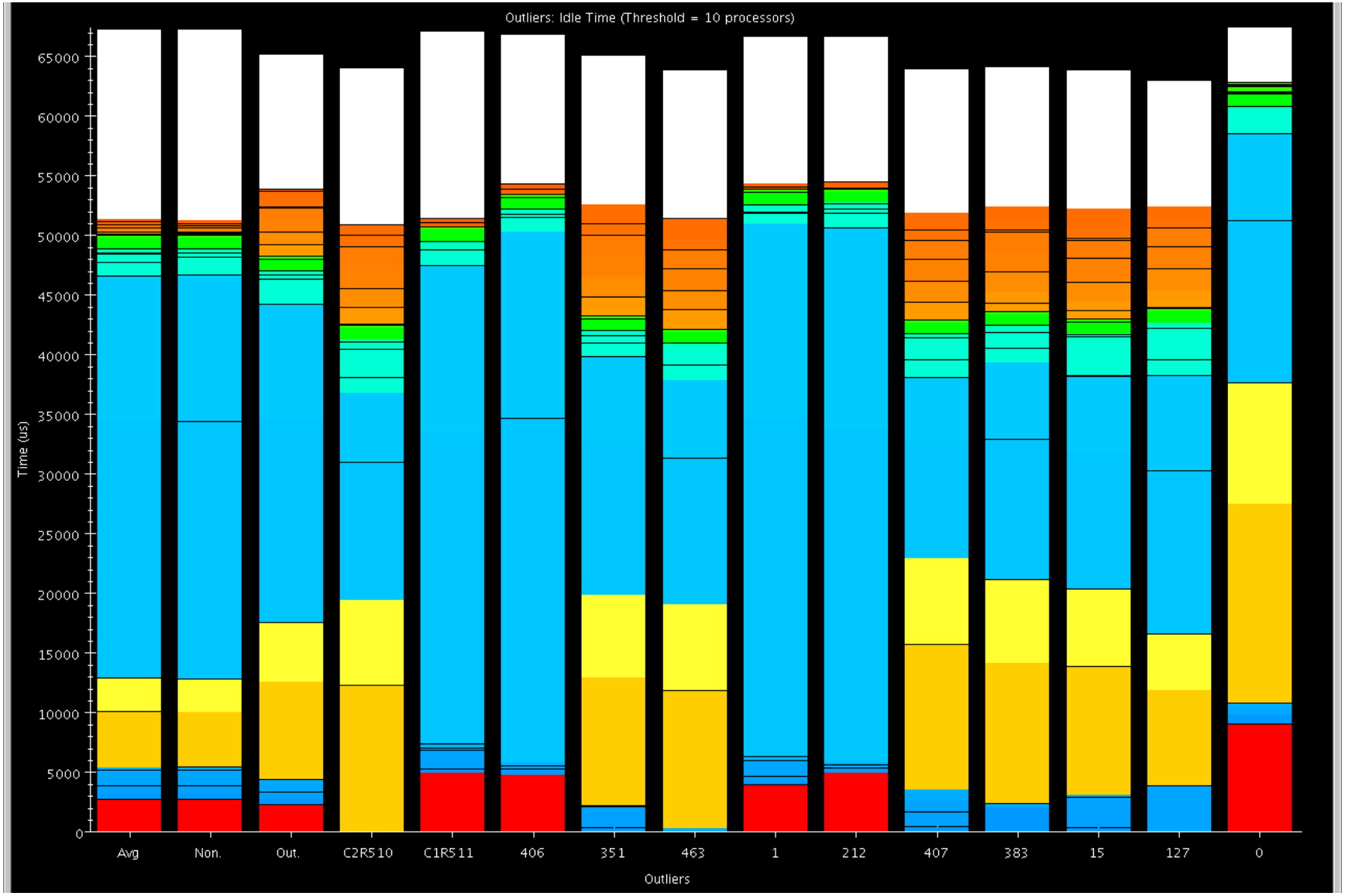


Clicking on EPs traces messages, mouseover shows EP details.

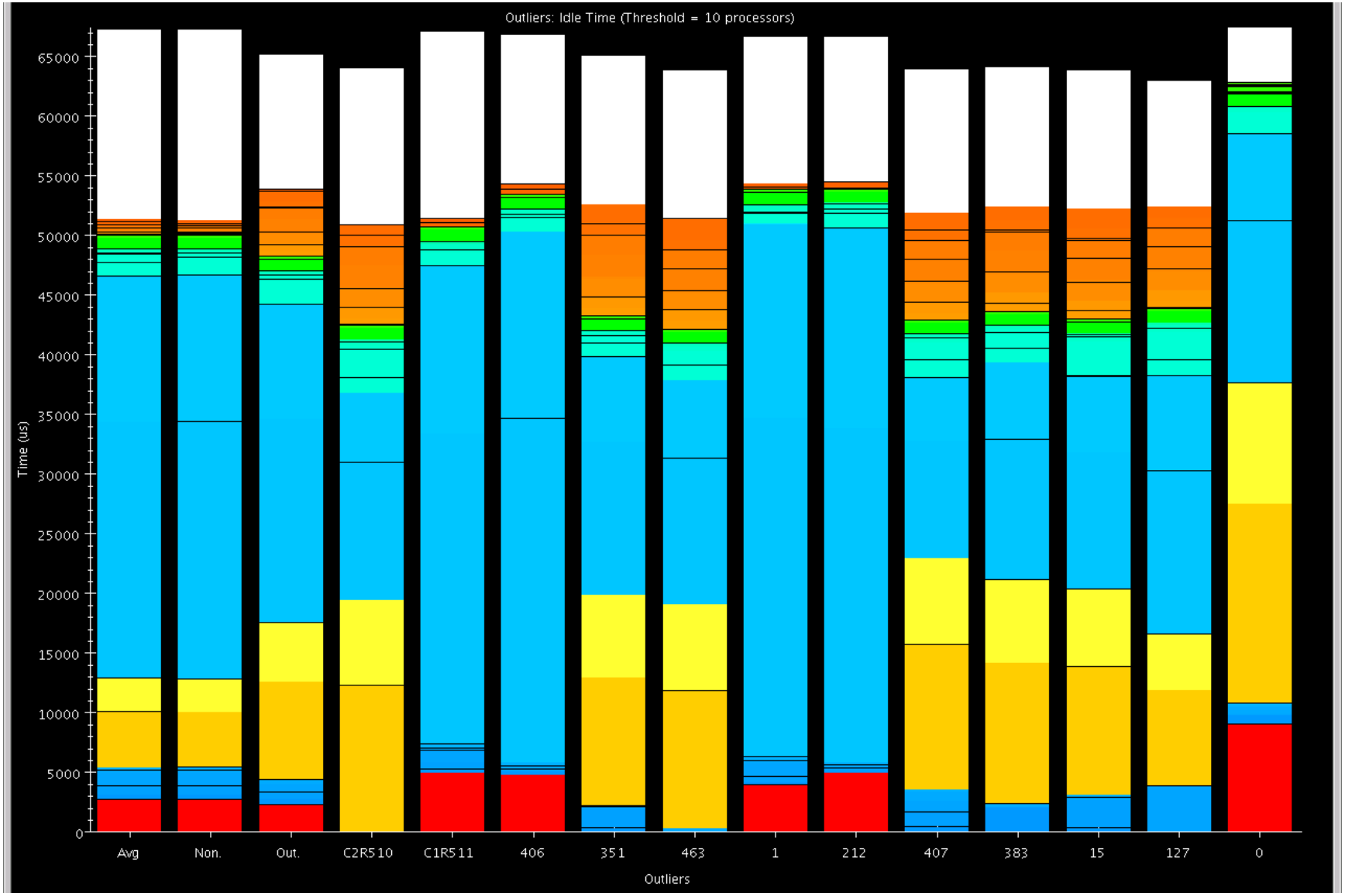


Colors are different EPs. White ticks on bottom represent message sends, red ticks on top represent user events.

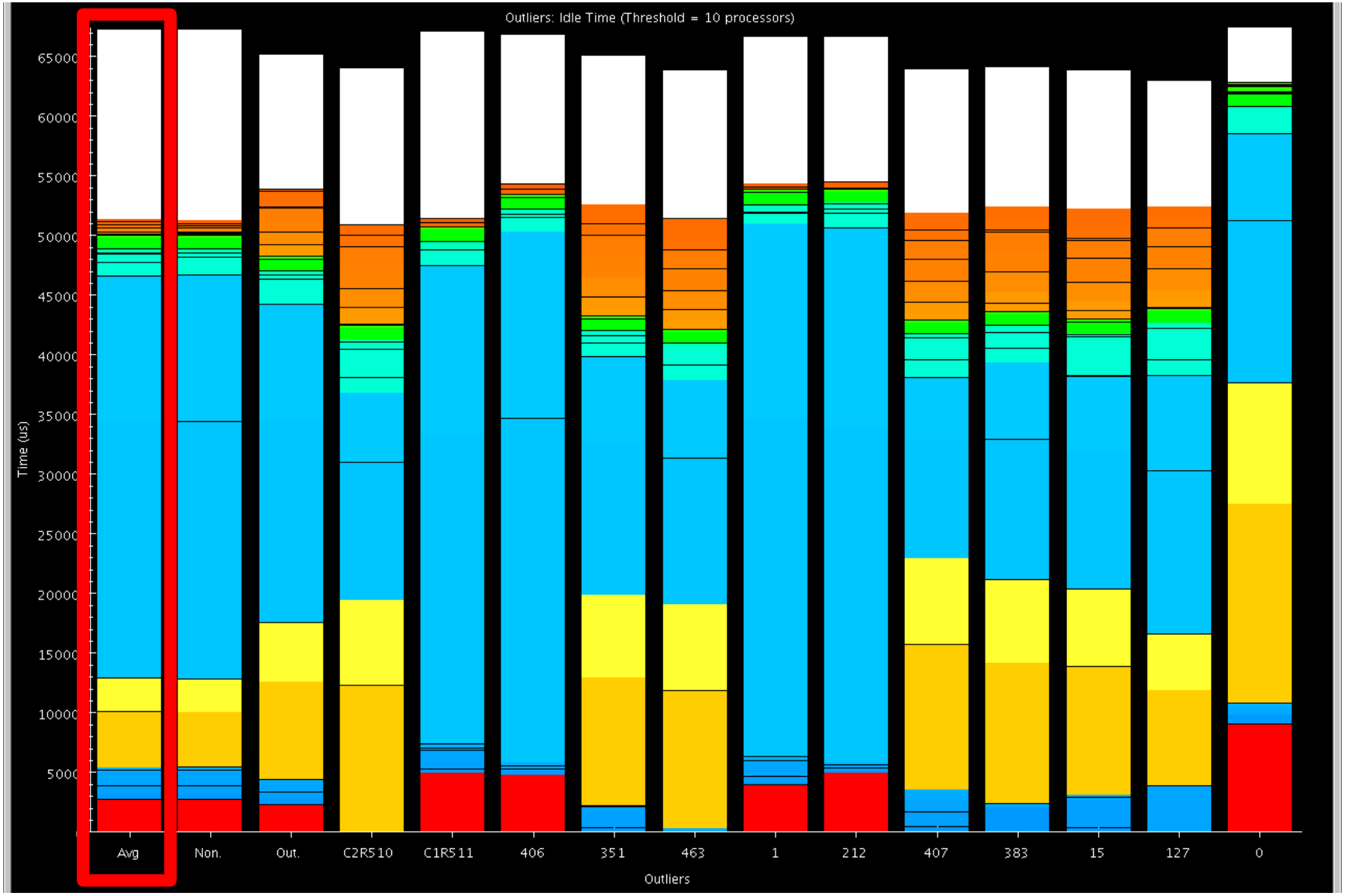
Processed Data Views



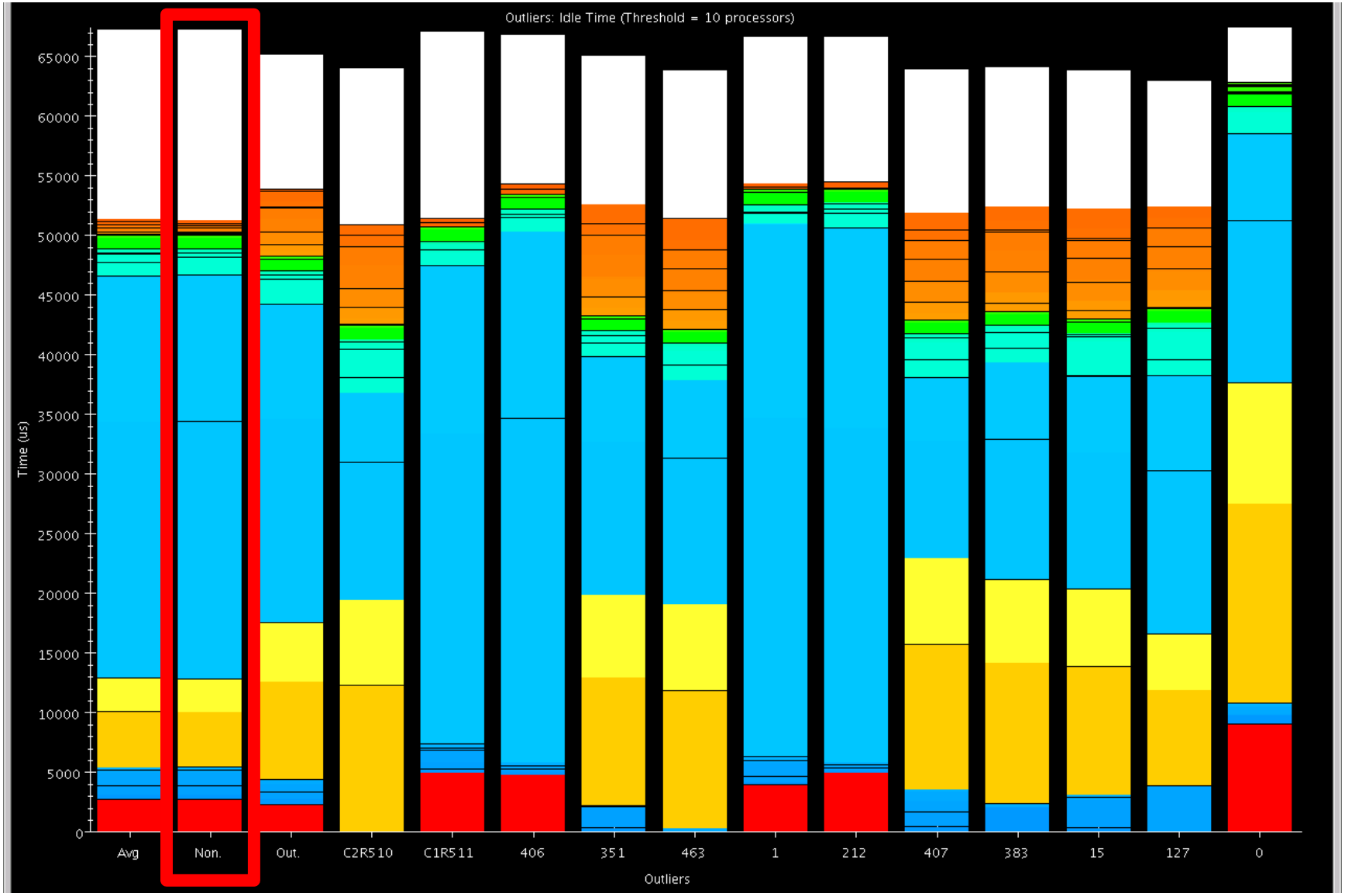
Outlier Analysis



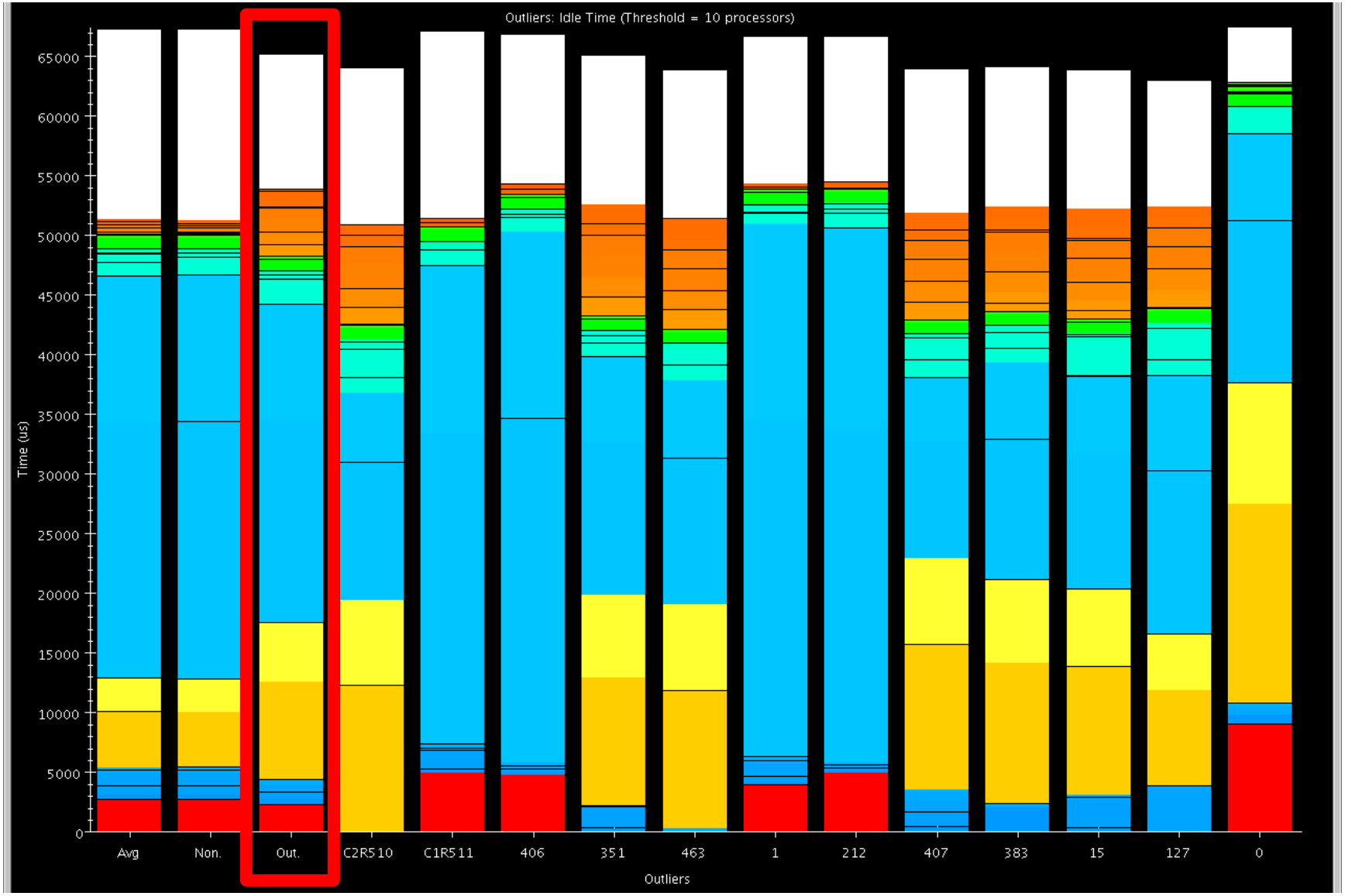
k-Means to find “extreme” processors



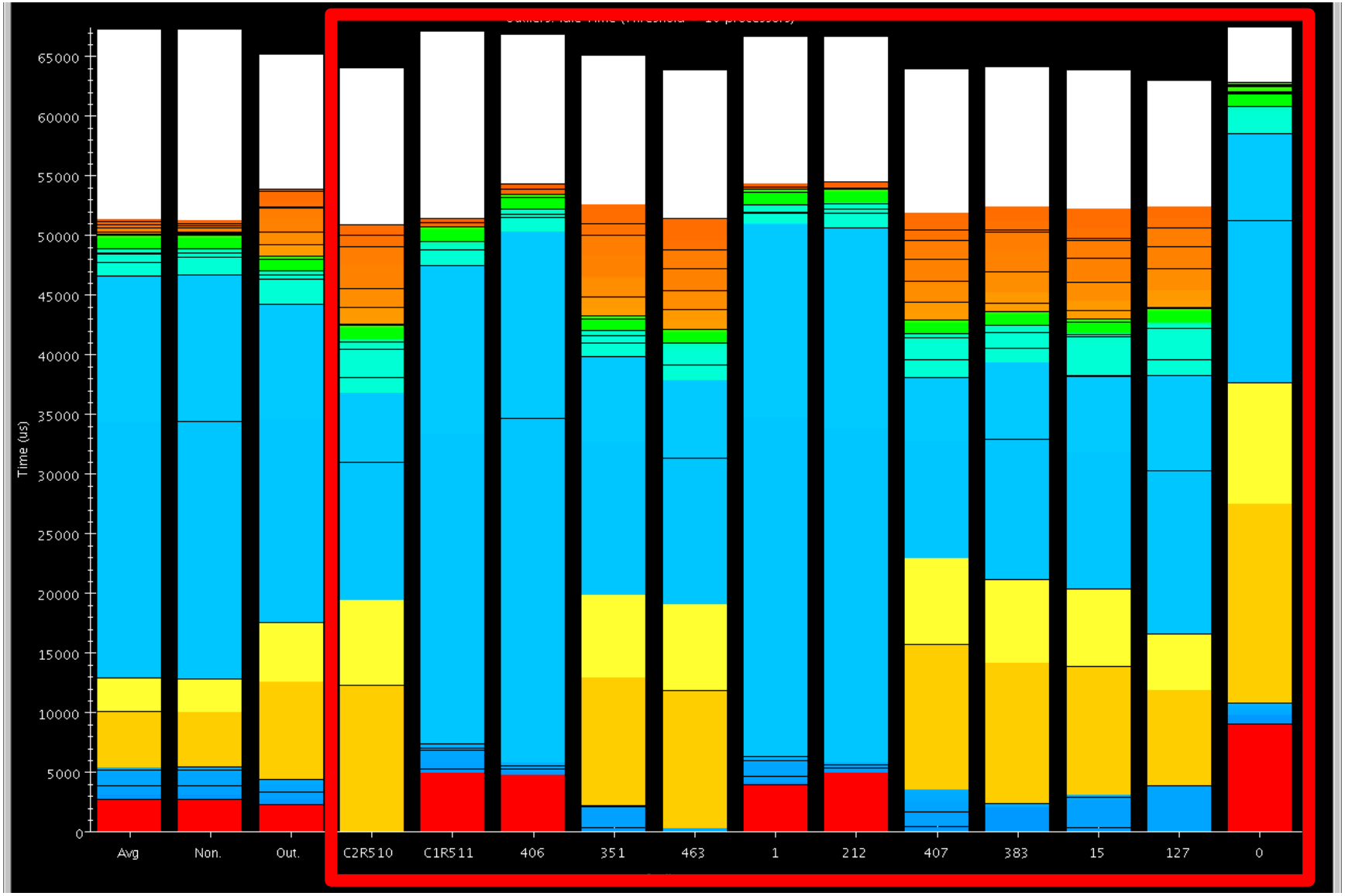
Global Average



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

Case Studies with Projections

Ronak Buch & Laxmikant (Sanjay) Kale
http://charm.cs.illinois.edu
Parallel Programming Laboratory
Department of Computer Science
University of Illinois at Urbana-Champaign

Basic Problem

- We have some Charm++ program
- Performance is worse than expected
- How can we:
  - Identify the problem?
  - Measure the impact of the problem?
  - Fix the problem?
  - Demonstrate that the fix was effective?

Key Ideas

- Start with high level overview and repeatedly specialize until problem is isolated
- Select metric to measure problem
- Iteratively attempt solutions, guided by the performance data

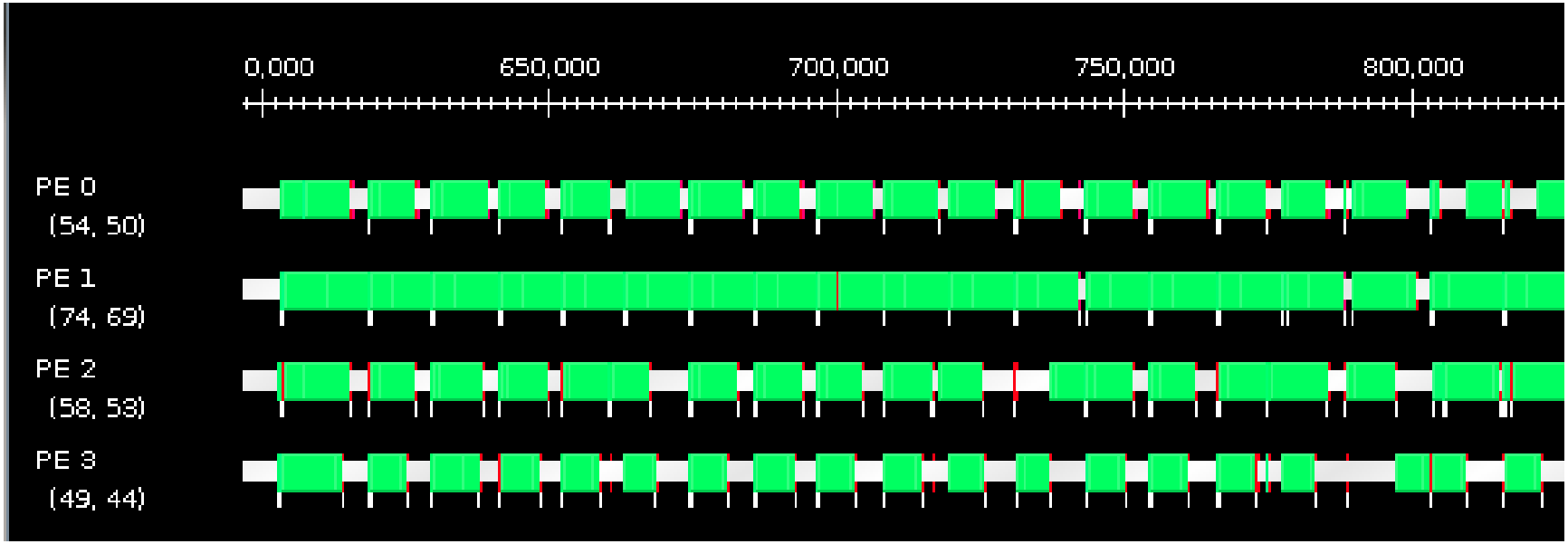
Stencil3d Performance

Stencil3d

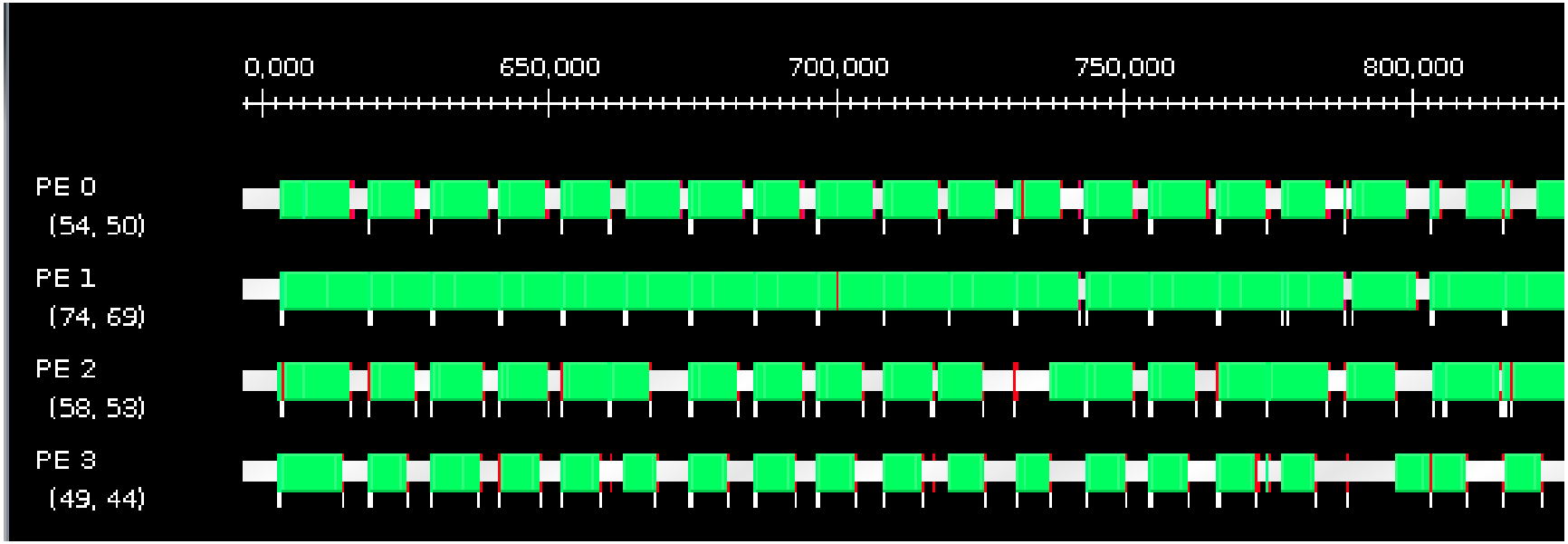
- Basic 7 point stencil in 3d
- 3d domain decomposed into blocks
- Exchange faces to neighbors

- Synthetic load balancing experiment
- Calculation repeated based on position in domain

No Load Balancing

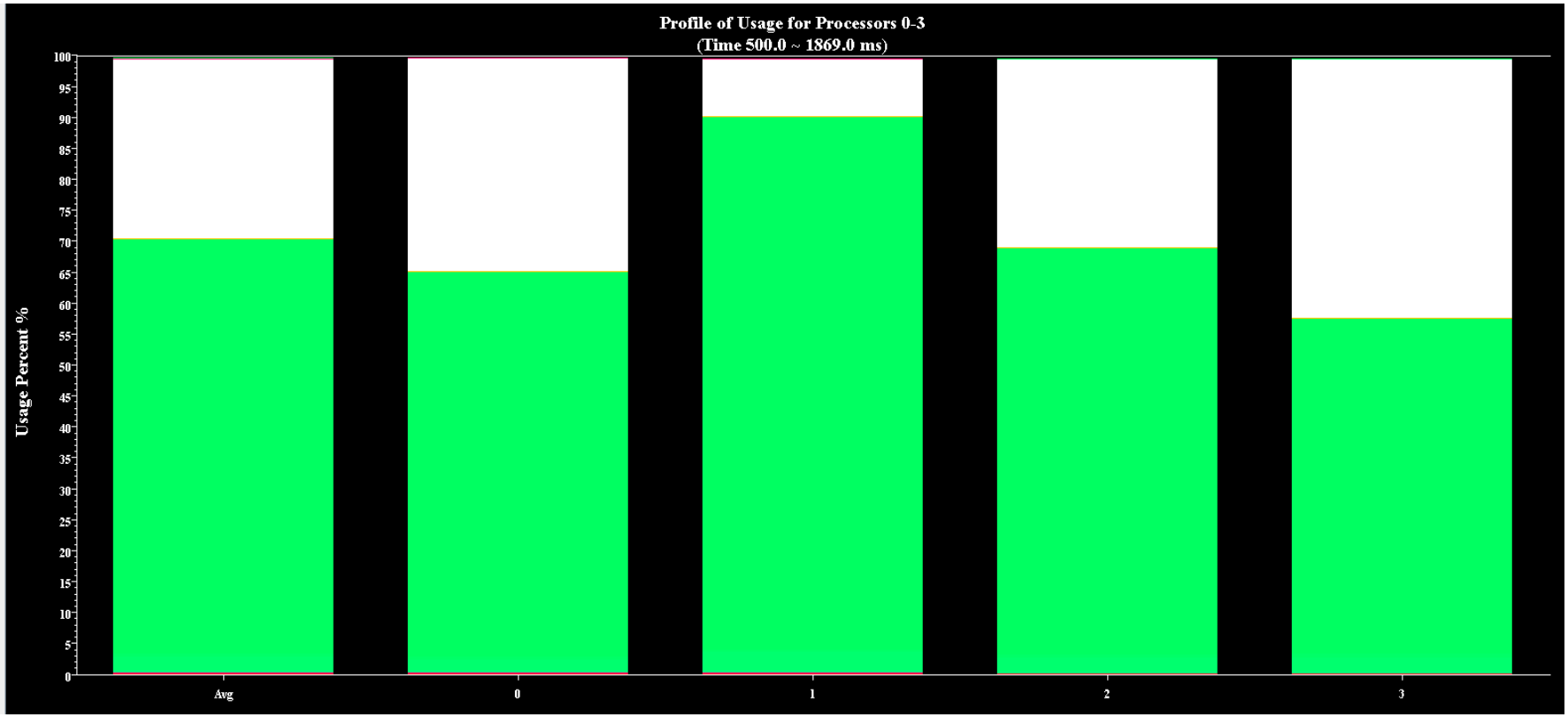


No Load Balancing



Clear load imbalance, but hard to quantify in this view

No Load Balancing

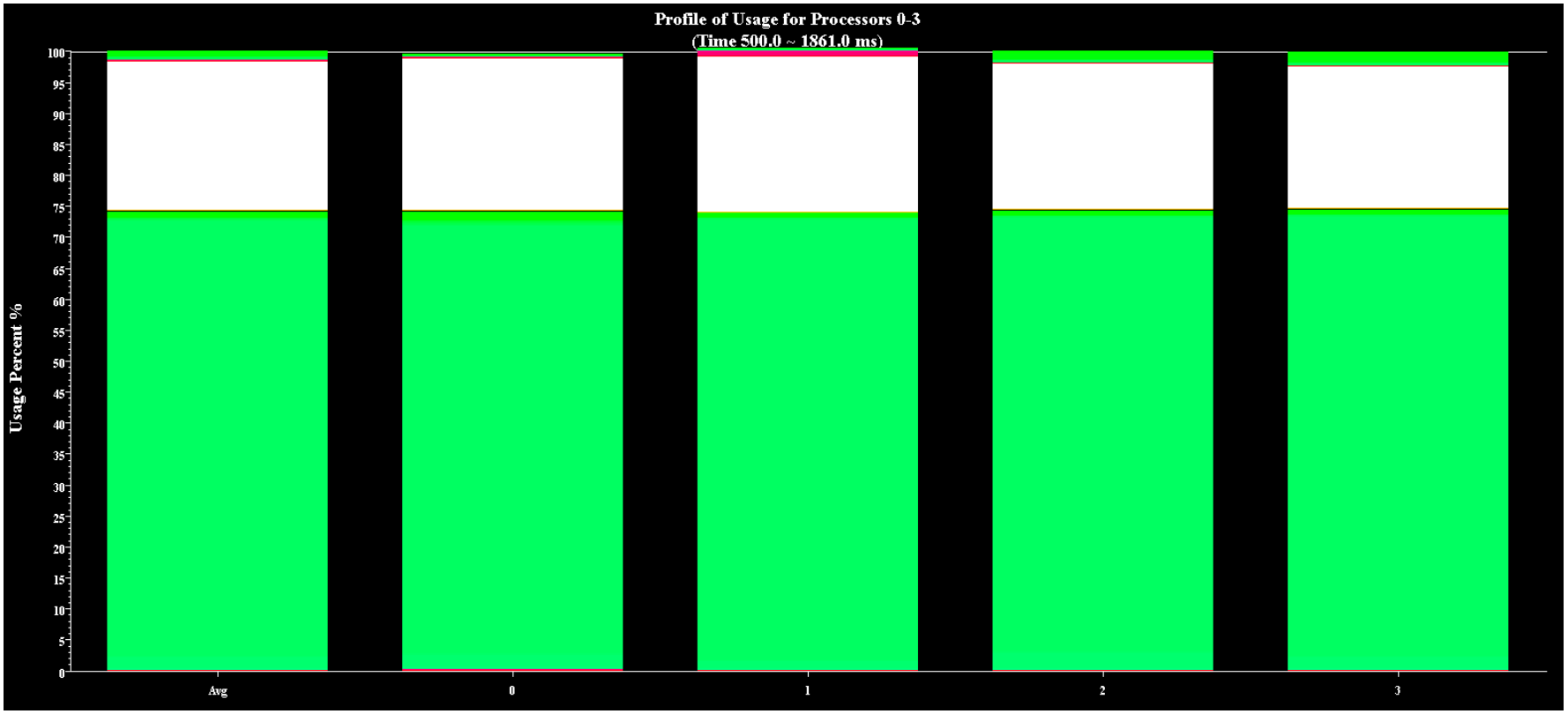


Clear that load varies from 90% to 60%

Next Steps

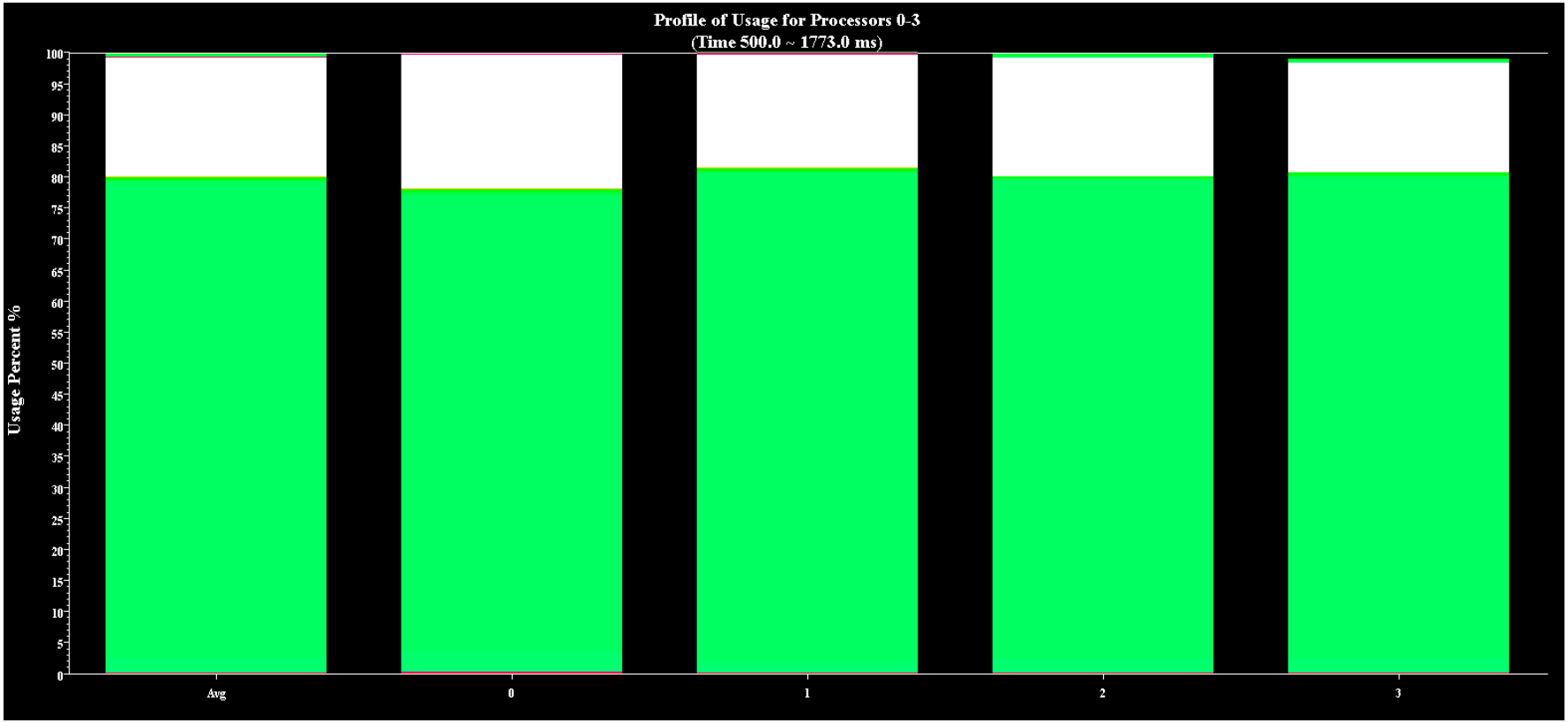
- Poor load balance identified as performance culprit
- Use Charm++'s load balancing support to evaluate the performace of different balancers
- Trivial to add load balancing
  - Relink using -module CommonLBs
  - Run using +balancer <loadBalancer>

GreedyLB



Much improved balance, 75% average load

RefineLB



Much improved balance, 80% average load

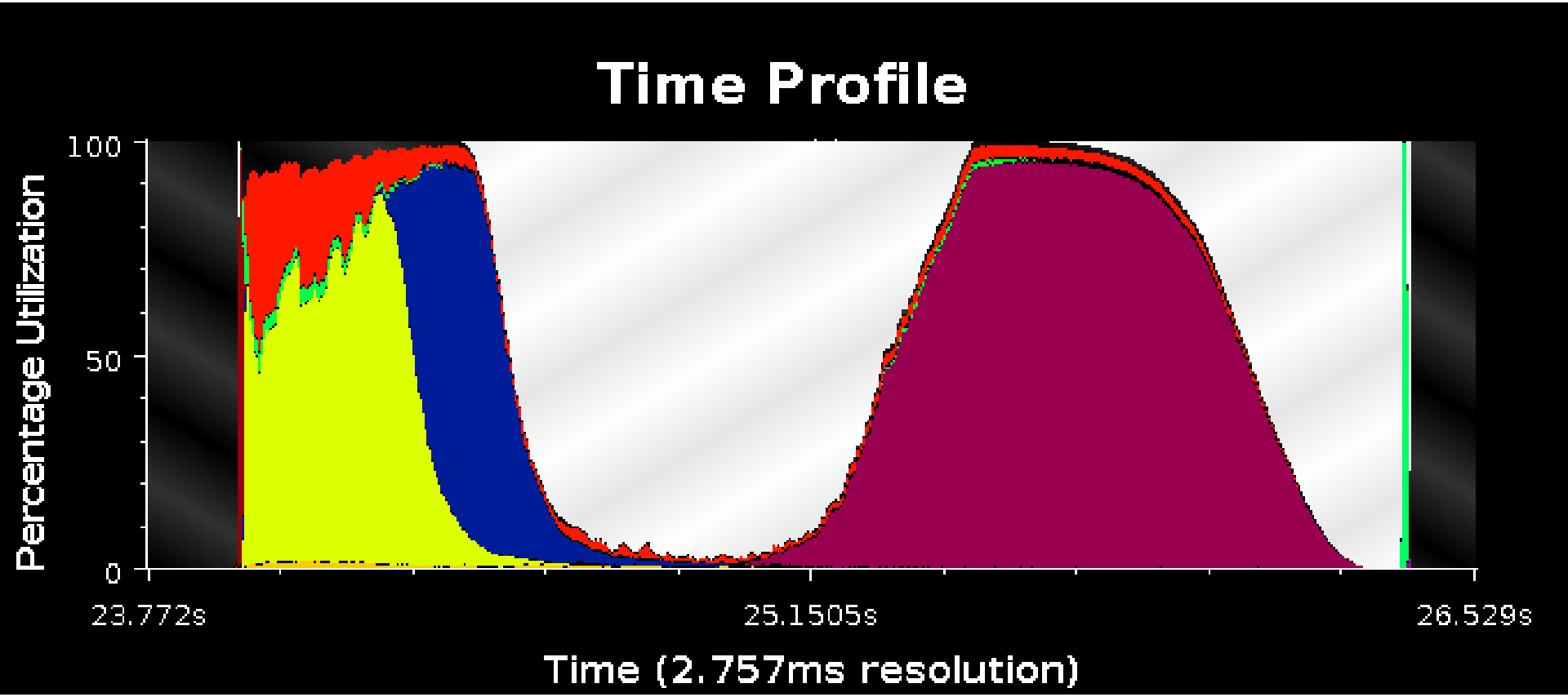
ChaNGa Performance

ChaNGa

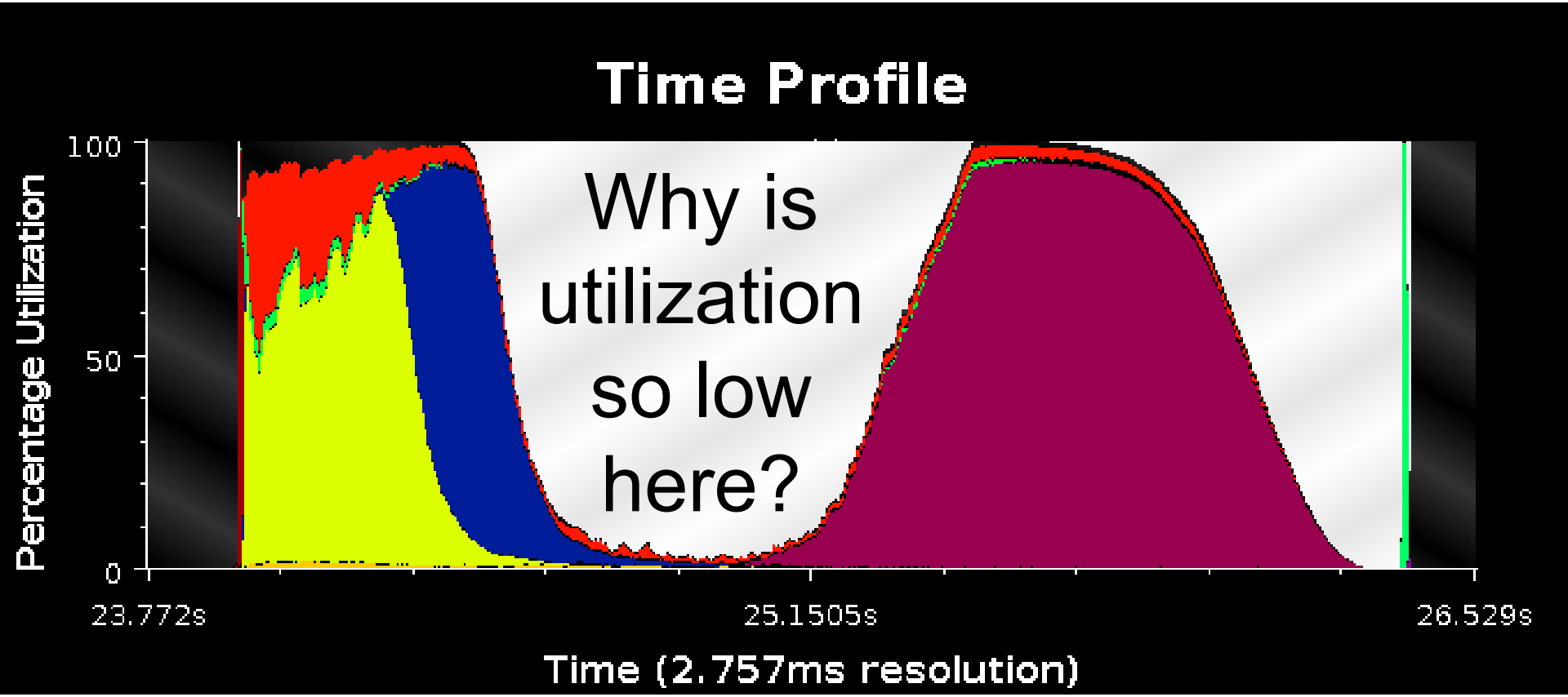
- Charm N-body GrAvity solver
- Used for cosmological simulations
- Barnes-Hut force calculation

- Following data uses *dwarf* dataset on 8K cores of Blue Waters
- *dwarf* dataset has high concentration of particles at center

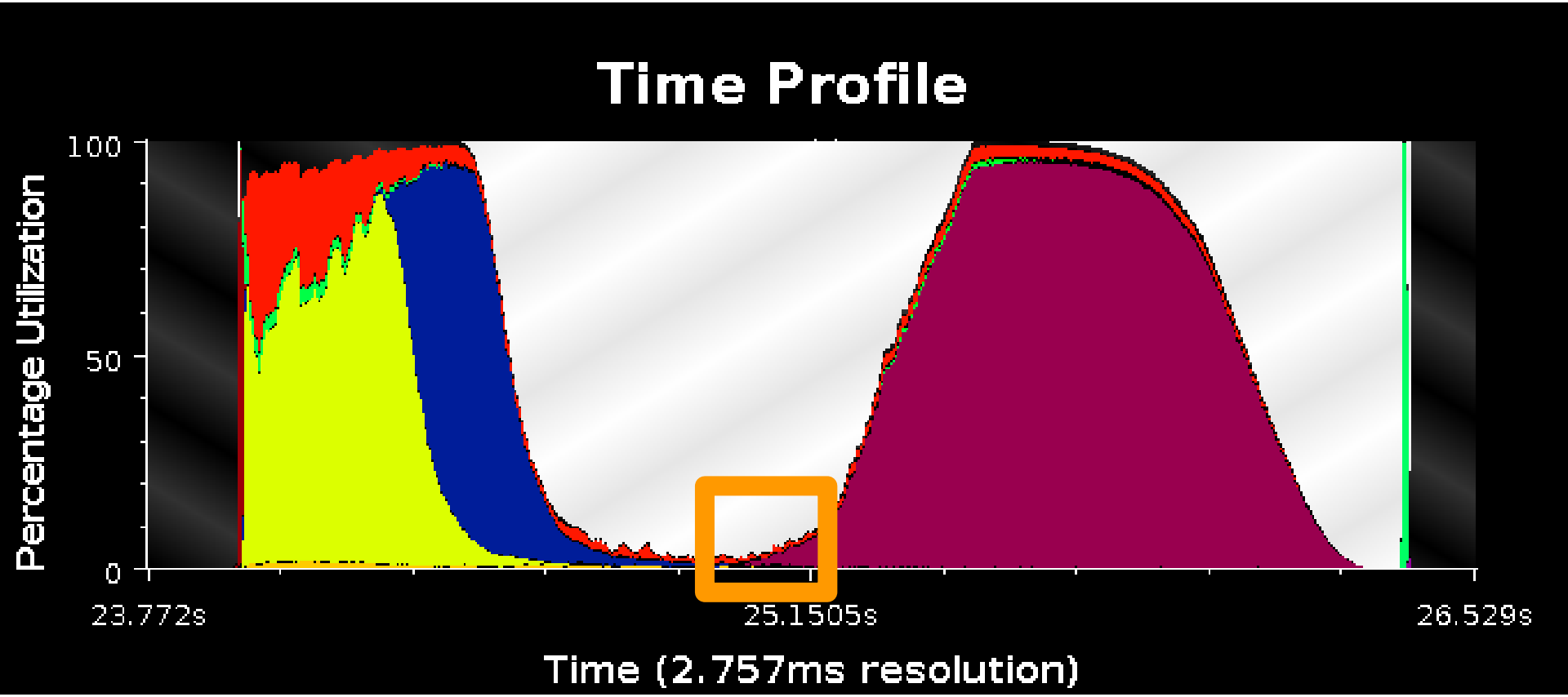
Original Time Profile



Original Time Profile



Original Time Profile

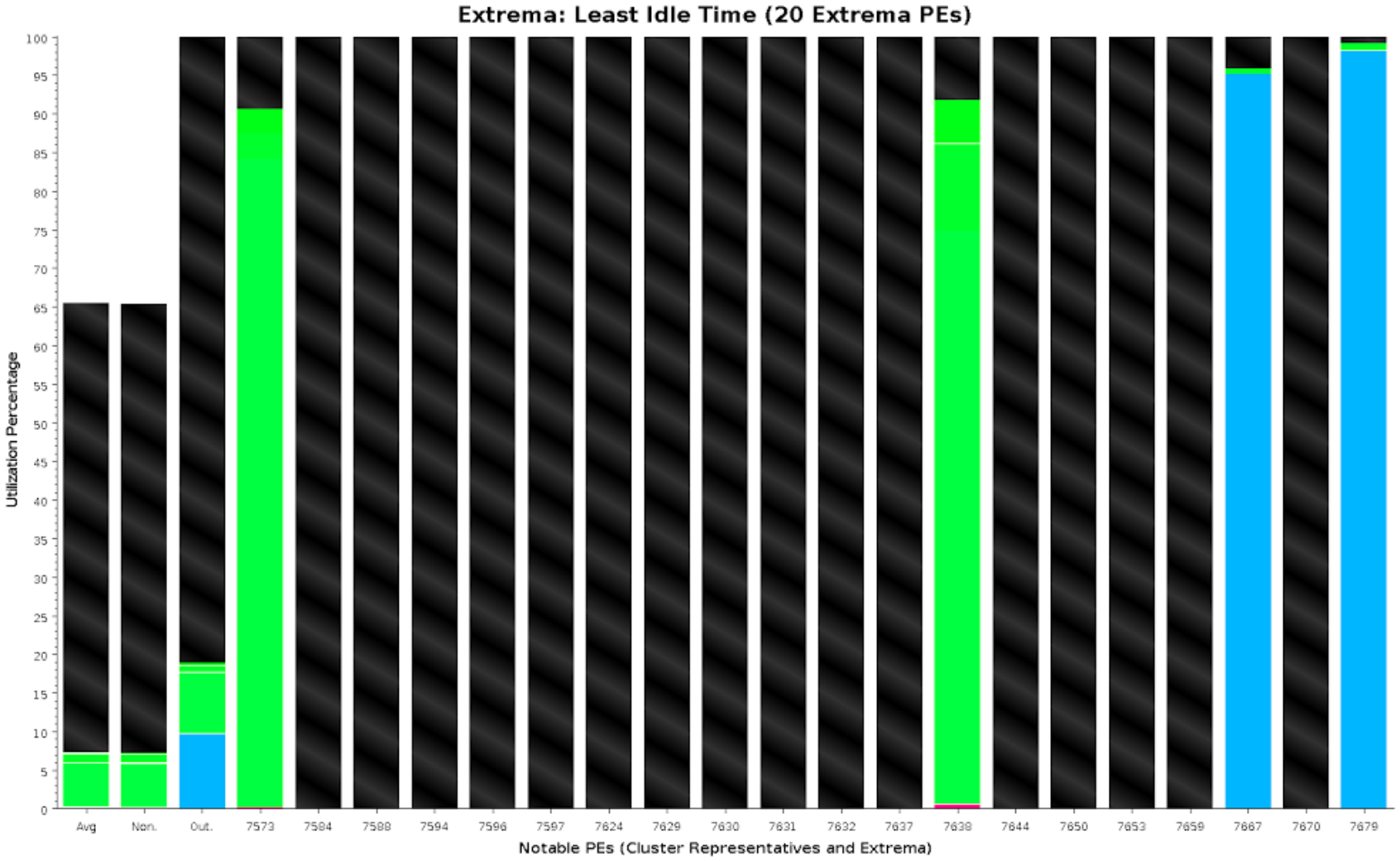


Some PEs are doing work.

Next Steps

- Are all PEs doing a small amount of work, or are most idle while some do a lot?
- Outlier analysis can tell us
  - If no outliers, then all are doing little work
  - If outliers, then some are overburdened while most are waiting

Outlier Analysis



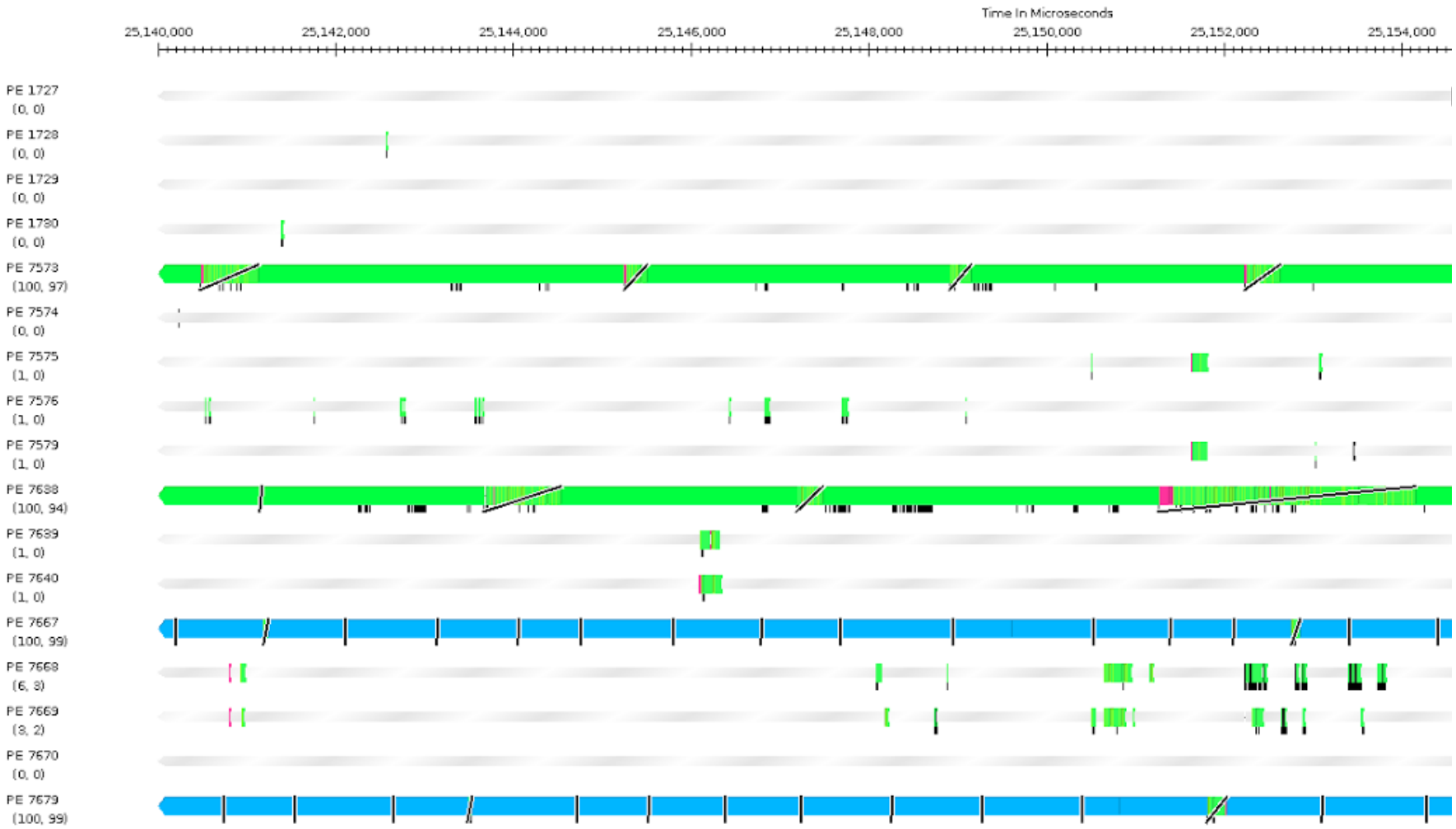
Outlier Analysis

Next Steps

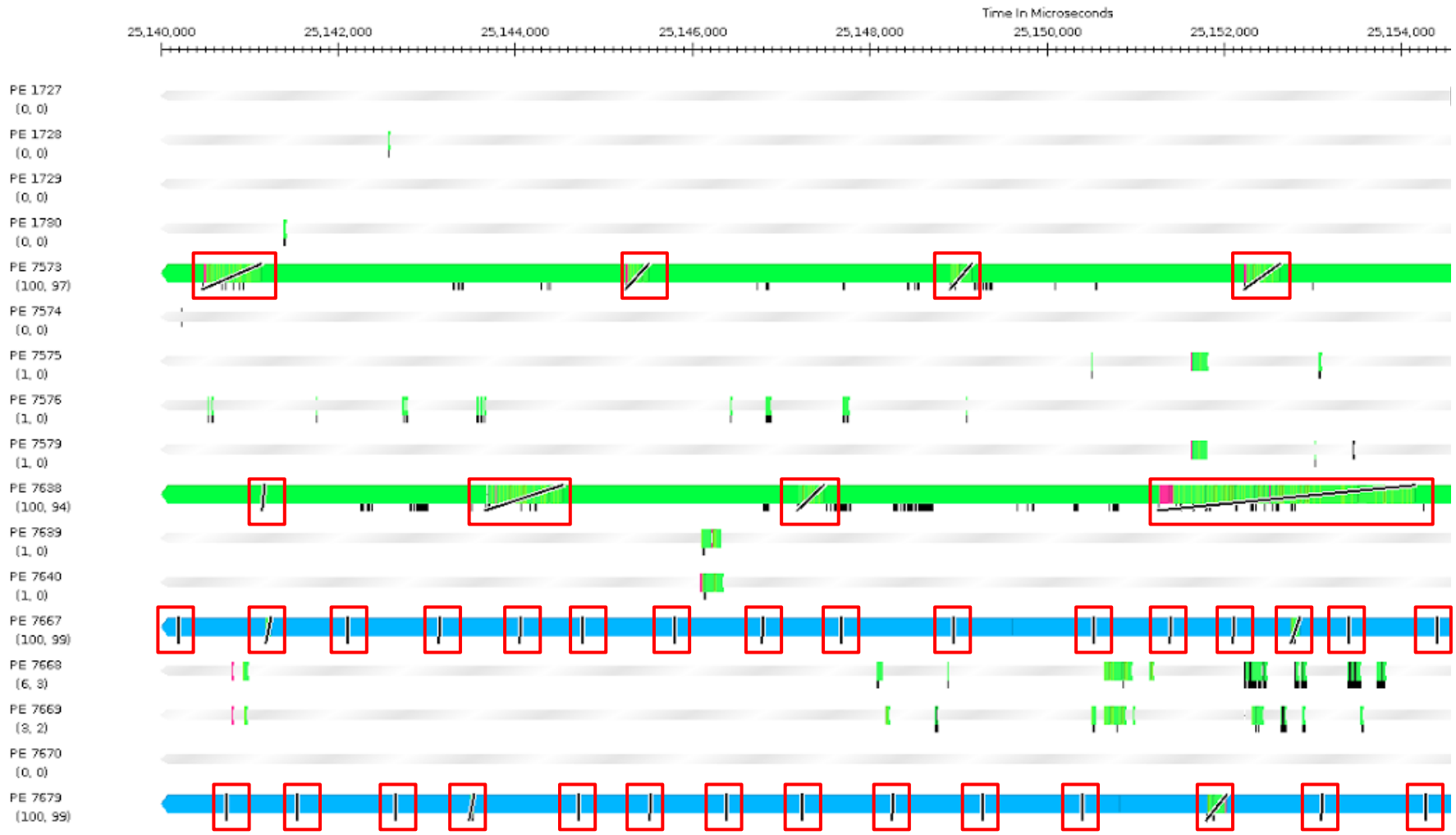
- Why does this load imbalance exist? What are the busy PEs doing and why are other waiting?

- Outlier analysis tells us which PEs are overburdened
- Timeline will show what methods those PEs are actually executing

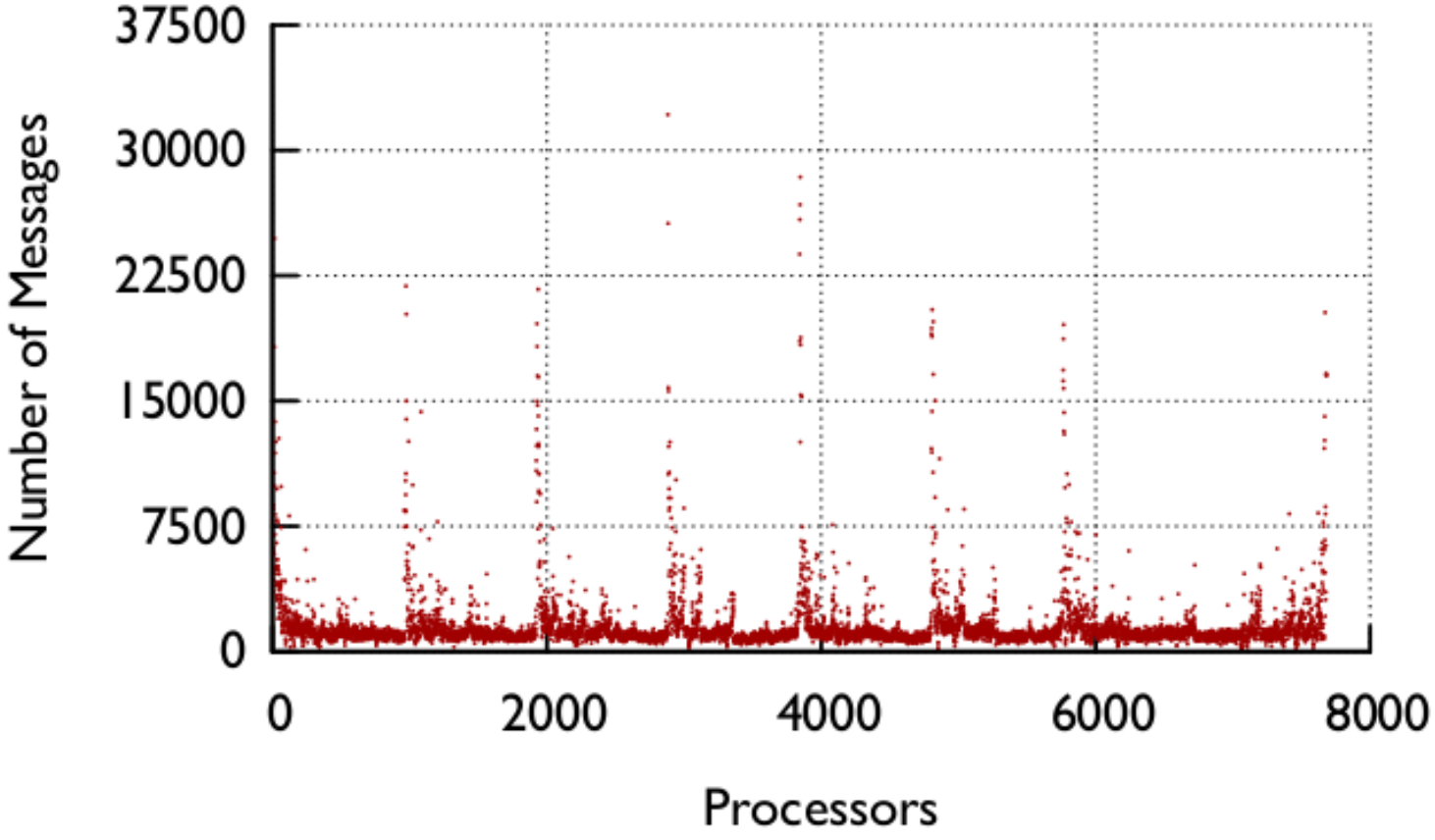
Timeline


Time In Microseconds
25,140,000
25,142,000
25,144,000
25,146,000
25,148,000
25,150,000
25,152,000
25,154,000
PE 1727
(0, 0)
PE 1728
(0, 0)
PE 1729
(0, 0)
PE 1730
(0, 0)
PE 7573
(100, 97)
PE 7574
(0, 0)
PE 7575
(1, 0)
PE 7576
(1, 0)
PE 7579
(1, 0)
PE 7638
(100, 94)
PE 7639
(1, 0)
PE 7640
(1, 0)
PE 7667
(100, 99)
PE 7668
(6, 3)
PE 7669
(3, 2)
PE 7670
(0, 0)
PE 7679
(100, 99)

Timeline


Time In Microseconds
25,140,000
25,142,000
25,144,000
25,146,000
25,148,000
25,150,000
25,152,000
25,154,000
PE 1727
(0, 0)
PE 1728
(0, 0)
PE 1729
(0, 0)
PE 1730
(0, 0)
PE 7573
(100, 97)
PE 7574
(0, 0)
PE 7575
(1, 0)
PE 7576
(1, 0)
PE 7579
(1, 0)
PE 7638
(100, 94)
PE 7639
(1, 0)
PE 7640
(1, 0)
PE 7667
(100, 99)
PE 7668
(6, 8)
PE 7669
(3, 2)
PE 7670
(0, 0)
PE 7679
(100, 99)

Original Message Count



Wrote new tool to parse Projections logs.

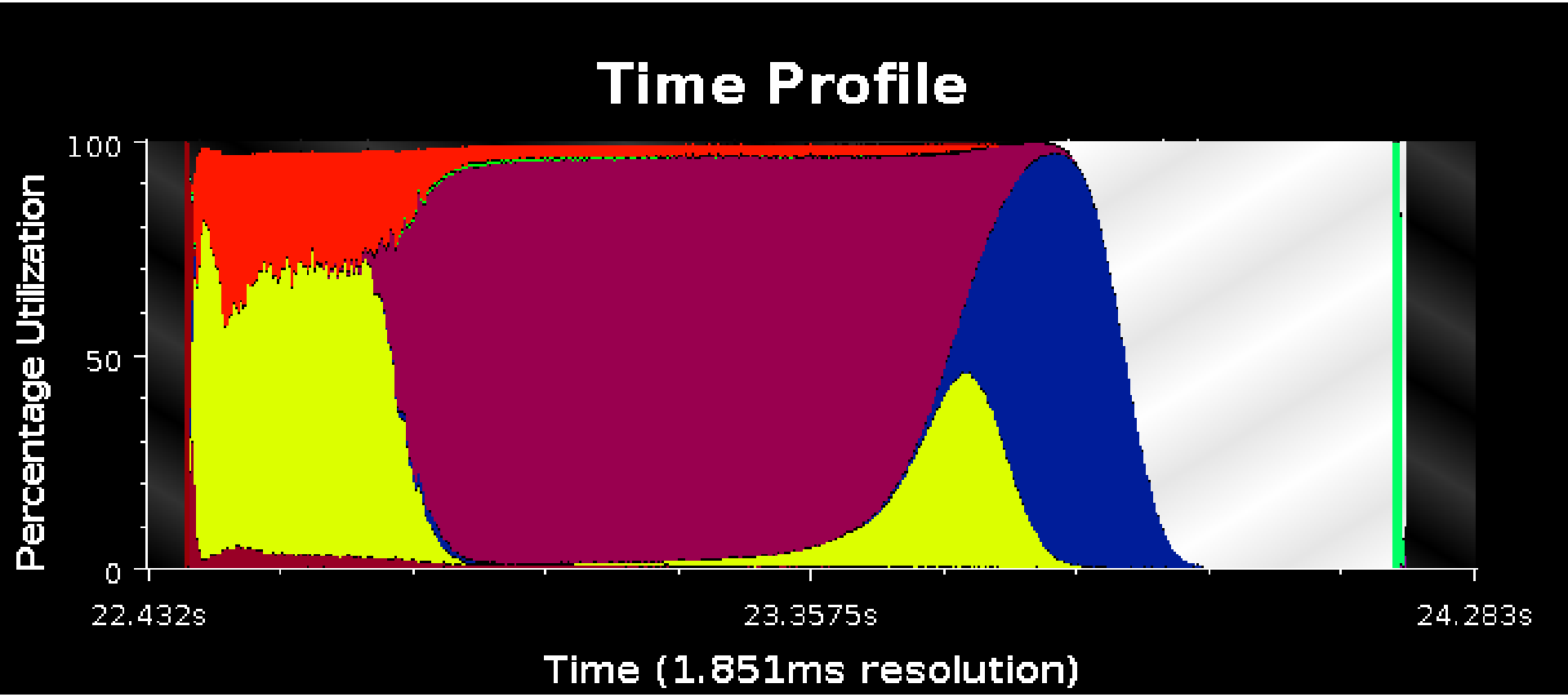
Large disparity of messages across processors.

Next Steps

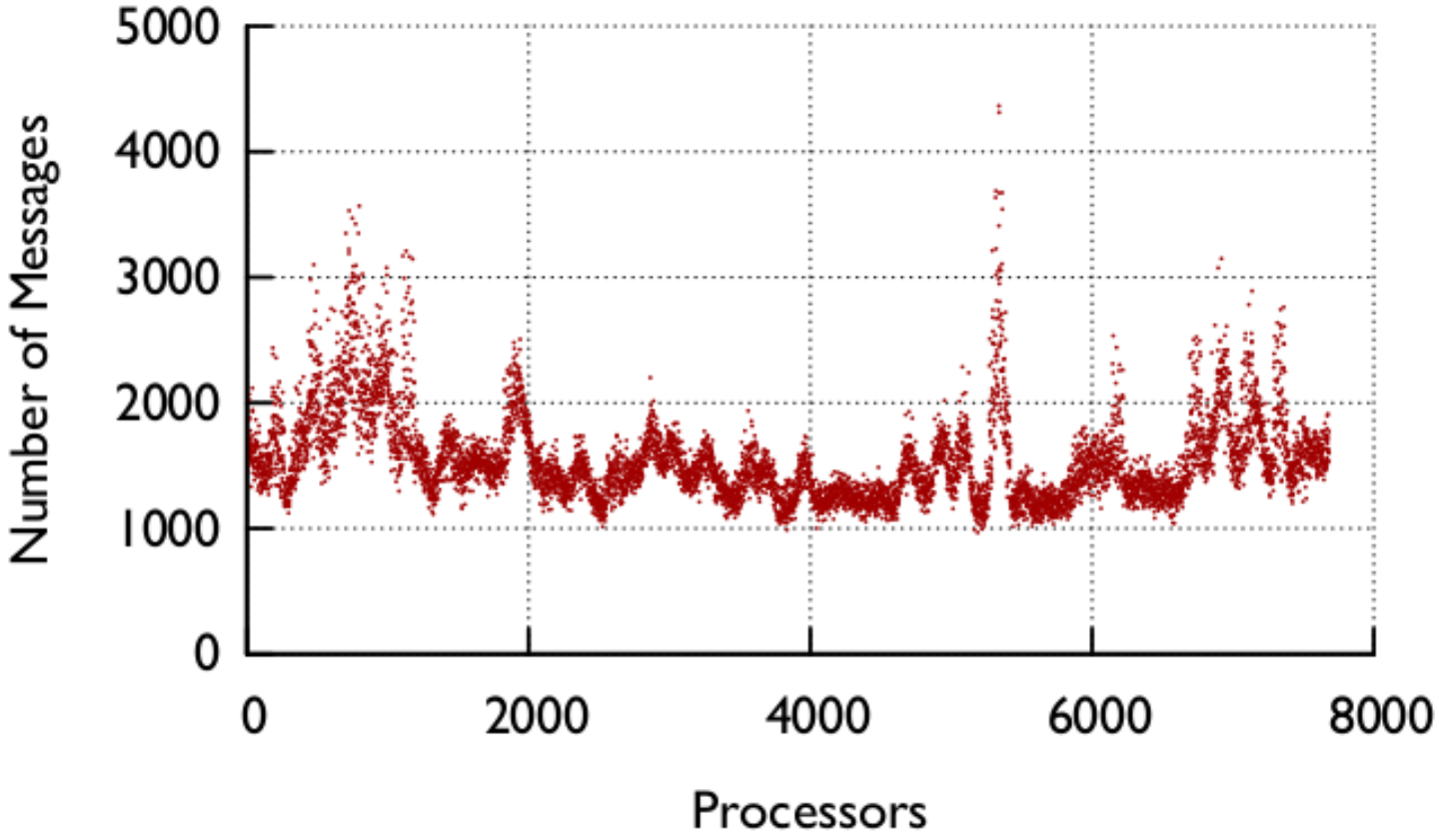
- Can we distribute the work?

- After identifying the problem, the code revealed that this was caused by tree node contention.
- To solve this, we tried randomly distributing copies of tree nodes to other PEs to distribute load.

Final Time Profile



Final Message Count



Used to have 30000+ messages on some PEs, now all process <5000. Much better balance.