Visualizing Multi-process CPU Utilization using CUSP

Mark W. Maimone
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA USA
Mark.Maimone@jpl.nasa.gov

***Abstract*— The CPU Utilization Statistics Plotter (CUSP) tool automates the interpretation of detailed CPU Utilization trace data and statistics. It puts you on the cusp of understanding how CPU resources are split among the many parallel components of a software system.**

CUSP combines time-sampled CPU utilization numbers and Event Log annotations to generate human-readable plots and tables. It automatically splits up large CPU usage log files around interesting events, determines and highlights just the tasks of primary relevance by evaluating their changing contribution to each plot's total CPU usage, automatically eliminates irrelevant tasks, provides context by labeling plots with names and durations of all active commands, and uses consistent color-coding to enable quick visual comparison across multiple plots.

CUSP has been used to process CPU Utilization trace logs on the Mars Science Laboratory and the Mars 2020 Rover missions during flight software development and Flight Operations on the Martian surface since December 2013.

TABLE OF CONTENTS

1. INTRODUCTION

Software systems with multiple parallel processes can be difficult to understand and debug. We wanted to understand how CPU time was being divided among multiple tasks, to give us the means of directly measuring unexpected delays in two realtime systems; the Curiosity and Perseverance Rover Flight Software (FSW) on the Mars Science Laboratory (MSL) and Mars 2020 (M2020) missions.

Curiosity's flight software is an embedded system running the VxWorks realtime operating system. [6] The MSL and M2020 embedded FSW use VxWorks' `taskHookLib` to implement logging of every task switch that occurs in the VxWorks realtime system, as part of the Operating System Abstraction Layer (OSAL) and Resource Tracking Service (RTS) FSW modules. There are more than one hundred independent software modules that comprise the FSW, and more than one hundred corresponding VxWorks tasks in the running system. [1] The best way to begin to understand unexpected delays is to measure the CPU usage of each running task, to see where time is actually being spent.

But measurement of CPU usage in our realtime system presented us with several difficulties.

There are so many task context switches that the amount of raw data needed to record them all rapidly grows to be impractically large, with several megabytes needed to document one minute of operation. Our deployed system could never downlink so much data, and even our testbed systems had trouble processing the large amounts of data generated over more than a trivial duration.

Another issue was the manual interpretation of the large amount of trace data. The process of breaking up the logged data into more understandable pieces initially required having a person specify start and end times of every period of interest by hand. Given that we can run dozens or hundreds of commands per session, this unwieldy process rapidly became unworkable.

We also needed a way to zoom in on just the tasks that contributed to the activity being studied. Our system has over a hundred independent tasks, but trying to visually grasp tens of thousands of CPU usage samples spanning so many tasks was daunting. We needed a way to restrict attention to just the tasks of interest, by creating a tool to determine which tasks were most active (instead of pre-specifying a smaller set of tasks to visualize, which would fail to elucidate unexpected task names).

Finally, we wanted a way to quickly understand the interrelationships between plots from different commands. Our existing CPU usage data plots offered no context, and the existing plotting tools used an ever-changing color scheme that made comparing different plots needlessly challenging.

2. RELATED WORK

Many commercial operating systems provide a realtime Performance Monitor, or Task Manager, with a simple visualization of the current overall CPU usage. However, these tools typically do not support after-the-fact trace-based diagnosis or per-task CPU Utilization numbers, nor do they typically provide event annotation. Johansson and Saegebrecht [2] summarize some of the tools available to help with performance visualization in realtime systems, and they also describe their own viewer. But these all report a single CPU usage number, not one broken down by tasks.

The `htop` tool [4] supports visualizing multiple processes' usages all at once. But it is limited to UNIX-based systems,

978-1-6654-3760-8/22/$31.00

Sampling Rate	Duration (hours : minutes : seconds)
64.0 Hz	3:45
8.0 Hz	30:00
1.0 Hz	4:00:00
0.1 Hz	40:00:00

Table 1. Time-capacity of a single 73 Mbit file holding 14,400 664-byte samples including 64 Hz statistics (1 Mbit equals 128 Kbytes). Note that after being compressed, this file typically would only require 5.6 Mbits to transmit.

Sampling Rate	Duration (hours : minutes : seconds)
64.0 Hz	15:00
8.0 Hz	2:00:00
1.0 Hz	16:00:00
0.1 Hz	160:00:00

Table 2. Time-capacity of a single 78 Mbit file holding 57,600 178-byte samples without extra statistics. Note that after being compressed, this file typically would only require 6.3 Mbits to transmit.

does not support a trace playback facility, and its use of the curses library for its display also prevents it from incorporating event-based annotations in a reasonable way (since there is little remaining screen area for such information).

VxWorks' spyLib provides CPU utilization statistics, and its WindView display tool can be used to for graphical display of some system state. But no known tools allow for simultaneous display of per-task CPU usage with event annotations, or for playback of trace data.

3. Example Realtime CPU data: MSL FSW

CUSP can generate plots using any well formed input files. But this section will describe the format of data available from the primary source used during its development, the MSL flight software.

MSL Resource Tracking Service FSW (RTS) can be instructed to log either an entire task context switch trace, or just the overall CPU Utilization numbers. The CPU Utilization data is generated onboard by monitoring all task context switches, but that detailed data is summarized and compressed down into statistics at one of four available sampling rates: 64 Hz, 8 Hz, 1 Hz, 0.1 Hz.

The per-task CPU Utilization (i.e., total duration of each task divided by the chosen sampling interval) is computed onboard, and stored as an 8-bit integer representing the percentage between 0 and 100 before being written into a file for transmission. In addition, the CPU Utilizations of all the constituent 64 Hz samples over the current sampling interval can optionally be summarized and reported as an 8-bit mean and 16-bit variance. These statistics enable additional insight into the finer-grained behavior of the system, at relatively little cost in generated data volume.

Sampling Rate	Raw size (Mbits)	92% Compressed size (Mbits)
64.0 Hz	1167.2	89.8
8.0 Hz	145.9	11.2
1.0 Hz	18.2	1.4
0.1 Hz	1.8	0.1

Table 3. Storage required for one hour of CPU Utilization data including 64 Hz statistics.

Sampling Rate	Raw size (Mbits)	92% Compressed size (Mbits)
64.0 Hz	312.9	24.07
8.0 Hz	39.1	3.01
1.0 Hz	4.9	0.38
0.1 Hz	0.5	0.04

Table 4. Storage required for one hour of CPU Utilization data without extra statistics.

The contents of the logged data include the timestamp at the start of the current block, the average CPU Utilization of each task over the sampling interval, and if requested the mean and variance of the each task's constituent 64 Hz samples. In the MSL FSW, each record including the 64 Hz statistics requires 664 bytes, while each record without statistics requires only 178 bytes.

These data are highly compressible, and internal tests have demonstrated a 13X compression benefit can be realized using the available MSL FSW LZO compression capability [3], reducing onboard storage requirements by 92%. Tables 1, 2, 3, 4 explain how much data can be logged by the system at its available sampling rates.

Event Annotations

System Event Annotations are needed to provide some context for the CPU Utilization numbers. For MSL, we use the names of the currently active spacecraft commands (extracted from Command Event Reports) and their associated durations as labels, but any set of time-tagged data may be used.

4. Automatic Plot Generation

One of the key benefits of CUSP over the tools that preceded it is that it can generate human-understandable plots without requiring a human in the loop to hand-craft each plot. In this section we explain how CUSP achieves that capability.

Stacked plots vs Unstacked Plots

CUSP supports two methods of plotting CPU utilization: Stacked and Unstacked. The **Stacked** display generates a color-filled histogram, where each new task's contribution is stacked on top of the previous one (so that visually they all add up to 100%). See Figure 1 for an example. The **Unstacked** display is the more traditional line-graph way of plotting CPU usage. See Figure 3 for an example. You can ask CUSP to generate plots this way, but it can rapidly get unwieldy, since many task plots will likely overlap.

You must use the Unstacked display mode if you want to see the MSL-embedded 64 Hz mean and variance data. See

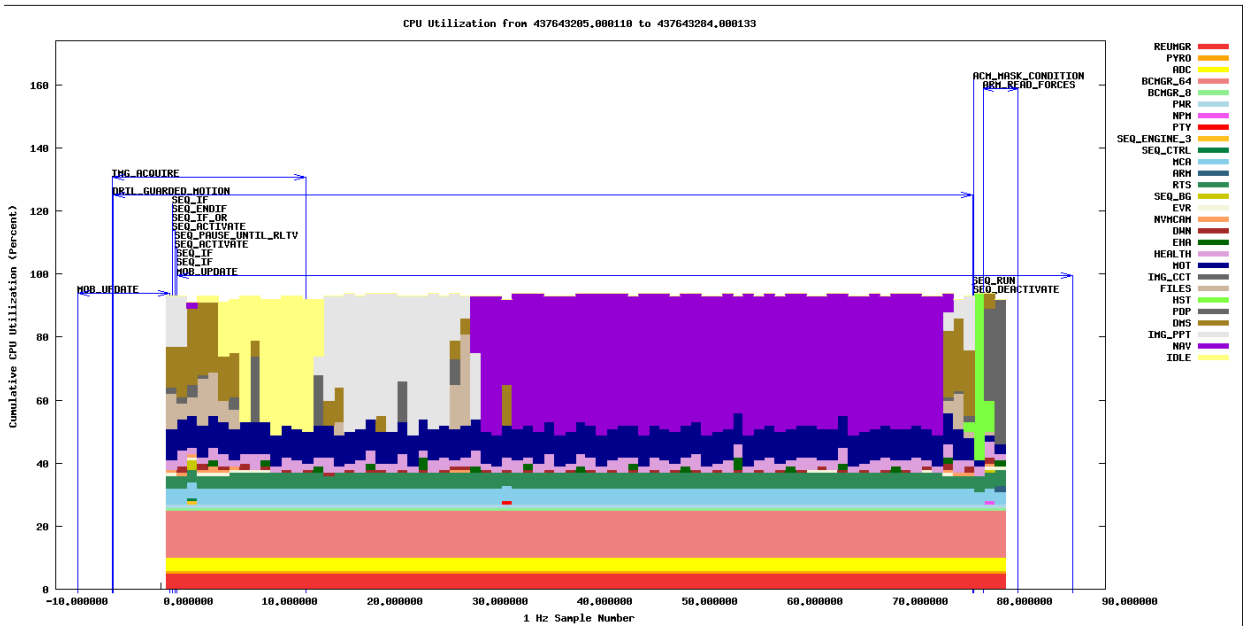

Figure 1. Example of a "Stacked" output plot. The bottom half of the plot shows those tasks whose CPU utilization remained relatively constant, the top half shows which task dominated the CPU for the given time duration.

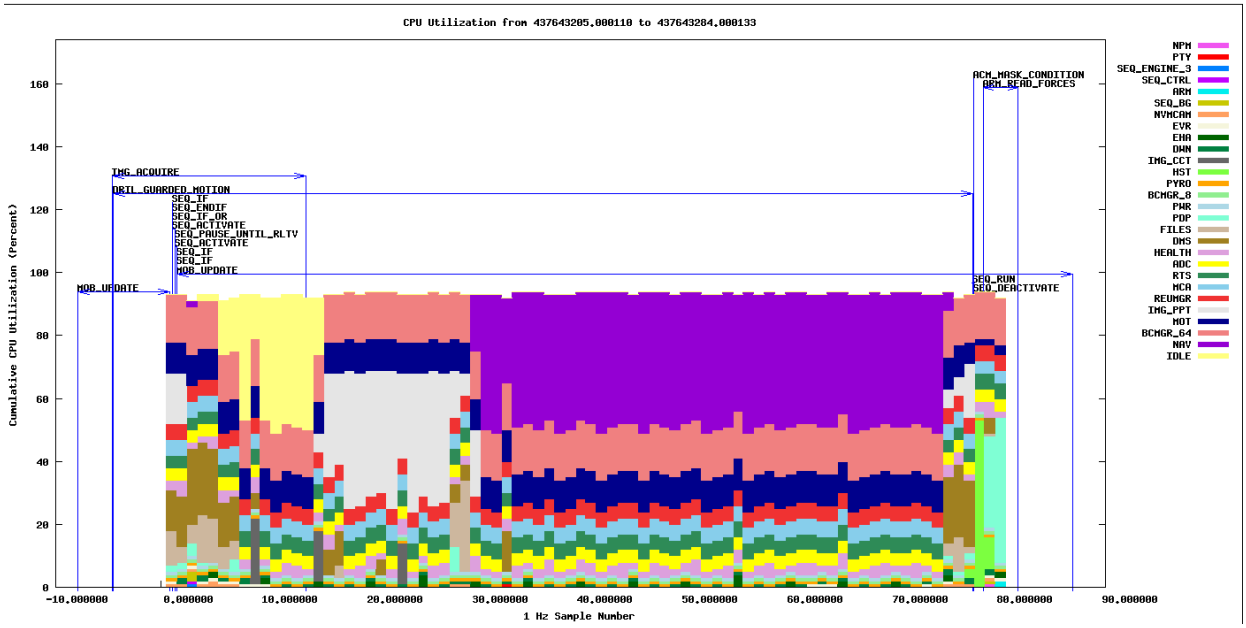

Figure 2. Illustration of the importance of smart task-ordering within the plot. This is the same plot as in Figure 1, but with tasks sorted by mean CPU utilization rather than by the variance. This plot is much harder to interpret at a glance.

Figure 4 for an example. It would be challenging to visualize CPU Utilization variance in the Stacked histogram display mode.

Time Range Selection

CUSP automatically breaks up very large log files into command-sized pieces, eliminating the need for human assistance in creating plots.

Commands are input to CUSP by specifying a file with a list of time-tagged Event Annotations (Command Names for MSL). All events are extracted and written as labels in the plots, alongside arrows representing their complete duration. See Figure 1 for an example.

Certain events serve a dual role as both label and plot separator. CUSP accepts an input regular expression; whenever an event's name matches that expression, CUSP will use its start and end times to break up the large log file into three pieces; before that command, during the command, and after that command. Each piece will be processed independently, and the before/after time ranges may be further split if additional commands of interest are found in their time ranges.

So every command of interest will get its own independent

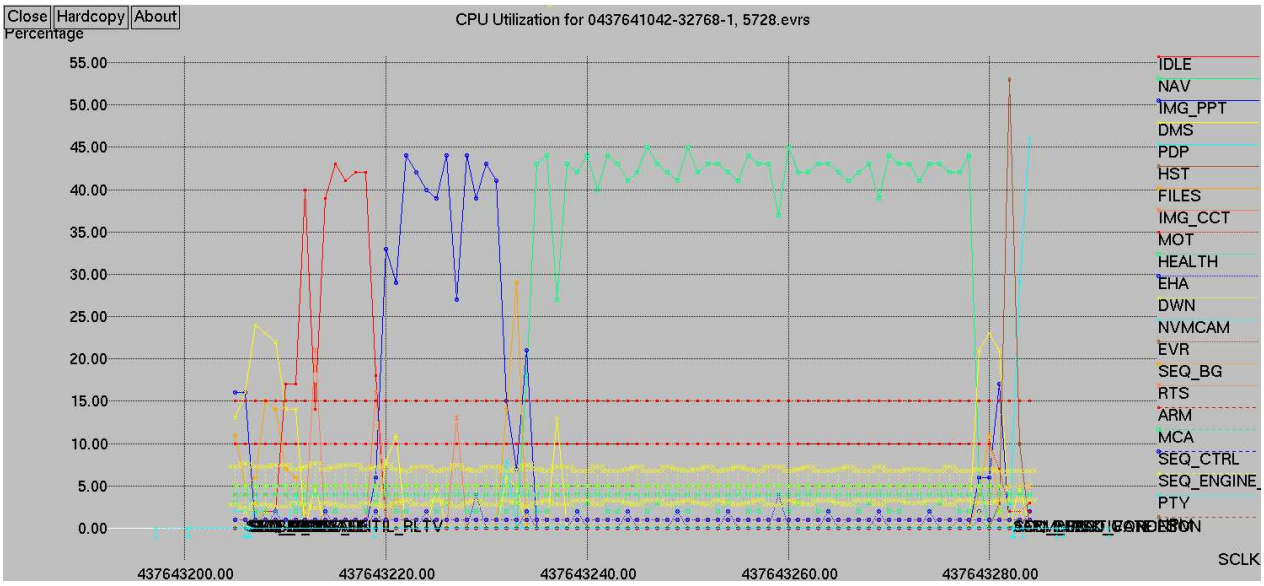

Figure 3. Example of an "Unstacked" output Plot. This is the same plot as in Figure 1, but with CPU usage plotted independently for each task. This plot is somewhat harder to interpret.

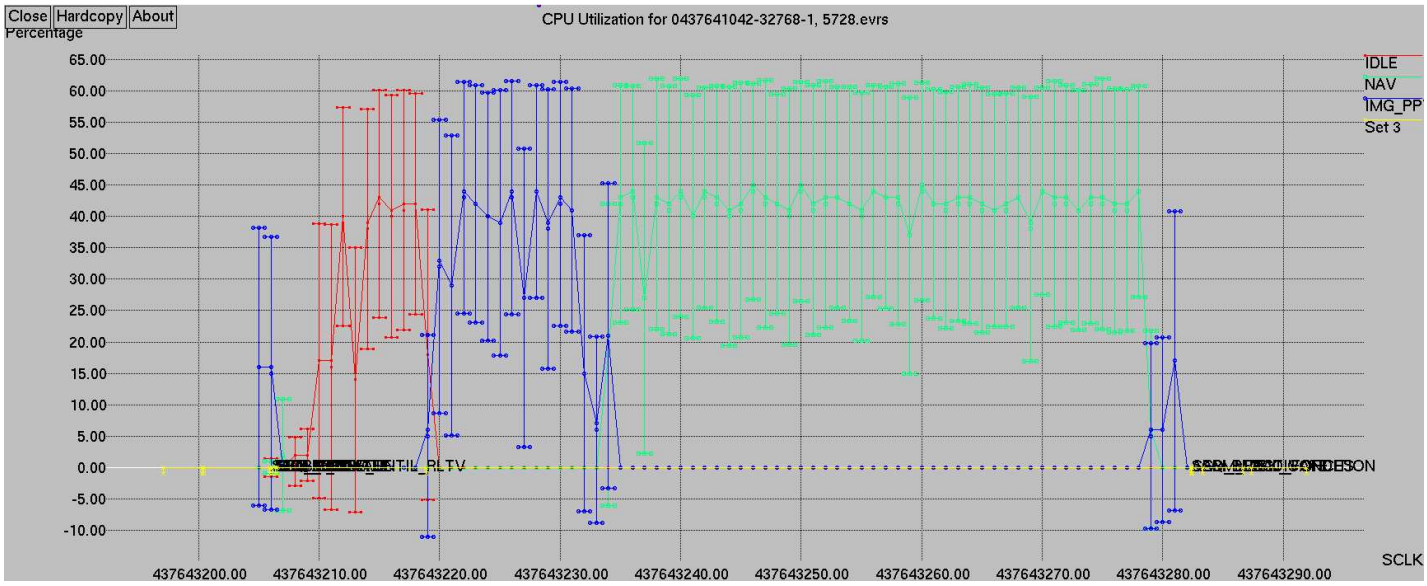

Figure 4. Example of an "Unstacked" output Plot with 64 Hz mean and variances. This plot in similar to Figure 3, but only includes three tasks on the command line so that the mean and variance can be seen as well.

plot, and the times surrounding each command of interest will also be written into separate plots. Humans need not bother with the tedious process of splitting up the complete CPU Utilization trace, CUSP will do it automatically.

Naming the Plots—Each plot is written to a file whose name includes both the name of the command used to determine its duration, and the current timestamp. These details allow the image filenames to be easily interleaved into other time-tagged system reports.

Task Ordering

Within each plot, the vertical axis represents Percent of CPU Utilization, and the horizontal axis represents the procession of time. One could imagine simply plotting the CPU usage of all the tasks in alphabetical order, but such a plot would be unreasonably difficult to understand when there are lots of tasks in the system. It would be difficult to read the names of the most important tasks in such a dense plot.

Although MSL (and Mars 2020) have over one hundred tasks, the important insight is that any interesting command will likely invoke a very small number of tasks over the entire command duration. So to help the reader focus in on where the CPU time is being spent, several factors are considered.

Eliminating unimportant tasks Only those tasks that have *at least one sample* with CPU Utilization above a fixed minimum will be included in the plot. Tasks that never reach a certain CPU Utilization are deemed insignificant, and will be omitted from the plot. CUSP computes this set of active tasks for you automatically, no need to guess at what they are

```
437643205.90076  ===  Command MOB_UPDATE completed
437643206.000133 Task IMG_PPT ran for  1.000 seconds with 16.000 +/-  0.000 percent CPU
437643206.10001  ===  Command SEQ_IF dispatched
437643206.10091  ===  Command SEQ_IF completed
437643206.10104  ===  Command SEQ_ENDIF dispatched
437643206.10167  ===  Command SEQ_ENDIF completed
437643206.10181  ===  Command SEQ_IF dispatched
437643206.10289  ===  Command SEQ_IF completed
437643206.10303  ===  Command SEQ_ACTIVATE dispatched
437643206.36621  ===  Command SEQ_PAUSE dispatched
437643206.36696  ===  Command SEQ_ACTIVATE completed
437643206.36861  ===  Command SEQ_PAUSE completed
437643206.36876  ===  Command SEQ_ACTIVATE dispatched
437643206.52307  ===  Command SEQ_IF dispatched
437643206.52368  ===  Command SEQ_ACTIVATE completed
437643206.52515  ===  Command SEQ_IF completed
437643206.52528  ===  Command SEQ_IF dispatched
437643206.52617  ===  Command SEQ_IF completed
437643206.52631  ===  Command MOB_UPDATE dispatched
437643210.000193 Task DMS     ran for   4.000 seconds with 21.500 +/-  2.693 percent CPU
437643213.000164 Task IDLE    ran for   3.000 seconds with 26.000 +/- 10.033 percent CPU
437643214.000161 Task IMG_CCT ran for   1.000 seconds with 39.000 +/-  0.000 percent CPU
437643218.85806  ===  Command IMG_ACQUIRE completed
437643220.000105 Task IDLE    ran for   6.000 seconds with 36.500 +/-  8.921 percent CPU
437643233.000093 Task IMG_PPT ran for  13.000 seconds with 36.615 +/-  8.553 percent CPU
437643234.000084 Task FILES   ran for   1.000 seconds with 21.000 +/-  0.000 percent CPU
437643235.000084 Task IMG_PPT ran for   1.000 seconds with 43.000 +/-  0.000 percent CPU
437643279.000114 Task NAV     ran for  44.000 seconds with 41.636 +/-  4.178 percent CPU
437643282.000090 Task DMS     ran for   3.000 seconds with 32.333 +/- 14.636 percent CPU
437643282.35684  ===  Command DRIL_GUARDED_MOTION completed
437643282.39595  ===  Command SEQ_RUN dispatched
437643282.43024  ===  Command SEQ_DEACTIVATE dispatched
437643282.43097  ===  Command SEQ_DEACTIVATE completed
437643282.43221  ===  Command SEQ_RUN completed
437643282.43236  ===  Command ACM_MASK dispatched
437643282.43341  ===  Command ACM_MASK completed
437643283.000101 Task HST     ran for   1.000 seconds with 29.000 +/-  0.000 percent CPU
437643283.34813  ===  Command ARM_READ dispatched
437643284.015758 Task PDP     ran for   1.016 seconds with 46.000 +/-  0.000 percent CPU
```

Figure 5. Text Table

by artificially restricting them on the command line.

Matching a pattern However, if the user really wants to, they may optionally request that only tasks matching a regular expression be included in the output.

Building a foundation Tasks whose CPU Utilization is significant, yet changes very little over time, are placed at the bottom of the plot. This gives the plot the appearance of a fence with a firm "Foundation", an easy visual cue that for the tasks on the bottom, things are not changing much. Tasks that have a large CPU variation over the duration of the plot are placed near the top. Such a plot makes it easier to notice changes in CPU Utilization over time (see Figure 1 for an example).

The Sky's the Limit The one exception to the "Building a Foundation" item is that the IDLE task is always plotted on top. Doing that consistently makes it easier to interpret the rest of the tasks as always being active. This mimics other tools' CPU Utilization display as a line graph, with empty CPU usage always on top.

The way to achieve the smooth "Foundation" appearance of the plots is to sort tasks using the *variance* of all their CPU Utilizations across the entire plot. This seems somewhat counterintuitive, in that one might think that placing tasks with large *mean* CPU Utilization on the bottom would be more clear. But that kind of plot is very hard to read as CPU usage changes over time; any constant-usage tasks stretch up and down across the whole graph, painting wide and uneven swaths that detract from the other changing tasks that are the really interesting ones. See Figure 2 for an example of a plot whose tasks are sorted by mean CPU Utilization.

So within each plot, CUSP automatically computes the variance of each task's CPU Utilization values, and plots active tasks in order according to their changing contribution. That makes a clear visual distinction between steady-state tasks on the bottom, and changing CPU usage tasks on top.

Coloring

Consistent coloring of individual tasks is important. It is the only way to ensure that one can quickly compare one or more plots against each other. The IDLE task must always be the same color, and the tasks that CUSP discovers to be most important should always have the same color.

So the user is encouraged to provide a consistent color scheme, mapping task names to individual colors. If no such mapping exists, then the plot tools will make their own choices. And since each plot is generated independently, the color choices will probably not be consistent across all graphs.

CUSP allows you to make plots more understandable by allowing you to choose fixed colors, but CUSP does not automatically determine the fixed-color scheme for you.

Statistic	Units	Curiosity	Perseverance
Sols with CPU usage data	sols	51% (1684 of 3292)	79% (203 of 257)
RP Sequences	number of sequences	2807	212
RP Sequence Durations	hours	0.746 +/- 0.659 (0.00 : 5.48)	0.744 +/- 0.665 (0.00 : 3.84)
Data product files	number of files	2662	362
Data product file sizes	Mbits	14.0 +/- 18.5 (0.130 : 72.9)	13.6 +/- 12.1 (0.956 : 65.4)

Table 5. Statistics related to the in-flight collection of CPU Utilization data on Curiosity and Perseverance, through November 9, 2021 (sols 3292 and 257, respectively). Data product file sizes are the uncompressed "raw" file sizes. Statistics are presented as "Mean +/- Standard Deviation (Min : Max)".

Event Label Annotation

CUSP also annotates each CPU utilization plot with all event labels (e.g., Command Names) and their graphical durations by drawing an arrow from their start to their end time. This makes it easy to understand just what the overall system is doing in each plot. The colorful CPU loading histogram may be the most compelling part of the plot, but you also need some visual cue to distinguish between hundreds of plots. So command names are plotted one over the other, and when the number of commands exceeds a predefined limit, they and their arrows wrap around vertically (see Figure 1). If you were to have too many commands executing one after another this kind of display would get too cluttered and become unreadable, but so long as the system is not flooded with commands this format has been found to work well. Each MSL command typically starts at least one second later than the previous one, making this a viable strategy.

This automatic labeling of events is another key feature of CUSP that enables it to generate human-readable plots without manual intervention.

Most Important Task Table

Finally, in addition to plots that are color-coded and event-annotated, CUSP also creates a text table summarizing just the events of interest and the tasks that made the greatest contribution to them over each time interval. Statistics are computed and reported for any task whose peak CPU usage spans a long contiguous time period.

This output makes for a nice summary report, explaining which tasks were most active for each command. See Figure 5 for an example.

5. MANUAL PLOT GENERATION

CUSP also supports generation of plots according to manually-specified constraints.

64 Hz Statistics You may choose display the 64 Hz statistics if they were included in MSL raw CPU Utilization data, but you must use Unstacked mode.
Time range You may specify a start and end time range, if you do not want CUSP to compute it for you at each command.
Task names You may specify a regular expression to limit the tasks being displayed, if you do not want to rely on CUSP's minimum CPU utilization criterion.
Image Format You may change the size and output format of the resulting images to anything that GNUPlot can support for output.

6. INTERPRETING THE DATA

All of the example plots in this paper (and the text in Figure 5 as well) come from data that was collected from the MSL engineering model rover, the Vehicle System Testbed (VSTB). We were evaluating the use of Visual Odometry (VO) processing [5] as a means of monitoring slip while running the drill and collecting images in parallel.

One question that arose was why the number of VO slip measurements was smaller than expected. The MOB_UPDATE command that implements it normally finishes in around 47 seconds, but we discovered it was sometimes taking longer and wanted to know why. So we looked into the plot that was automatically extracted by CUSP, shown in Figure 1,

The labels on top of Figure 1 indicate the names of commands and sequence directives that were executed on the rover. In the figure we see one MOB_UPDATE ending (the right arrow ends at time the command terminates), and another beginning (the left arrow just under the label). The command duration is some 86 seconds, much longer than usual. The bulk of the VO processing performed by the MOB_UPDATE command occurs in the Surface Navigation (NAV) FSW module, which is colored purple in the plot. During a normal execution such as that shown in Figure 6, we would expect to see the command start by acquiring an image using an "IMG" FSW module (light and dark grey) for up to 5 seconds, then the NAV module would perform its analysis and complete after 42 seconds on average. But Figure 1 shows something completely different. The command starts, but the CPU remains partly IDLE (light yellow) for 11 seconds. The reason is explained by looking at the command names at the top of the plot; an IMG_ACQUIRE command was running in parallel, taking pictures of the drill while it is in motion. That command's processing held a lock on an imaging resource, and did not complete until 10 seconds into the MOB_UPDATE. And even after it completed, the data collected by IMG_ACQUIRE was still being processing by IMG_PPT (Image Post-Processing, in light grey) for 18 seconds. Only once that image data had been fully processed and stored as a compressed data product into the flash filesytem (Data Management System (DMS) in beige, writing FILES in tan), could the new images begin to be processed by NAV, some 25 seconds later than expected.

The same information is also summarized in the text view shown in Figure 5, which highlights the task that uses the greatest share of CPU within each sampling interval. It also highlights the NAV, IMG, and DMS tasks with numeric CPU usage numbers and timestamps.

This ability to explain why the system is performing a certain way (in this case with longer delays between VO updates) is key in demonstrating to review boards why complex interactions between modules are still safe for use in flight

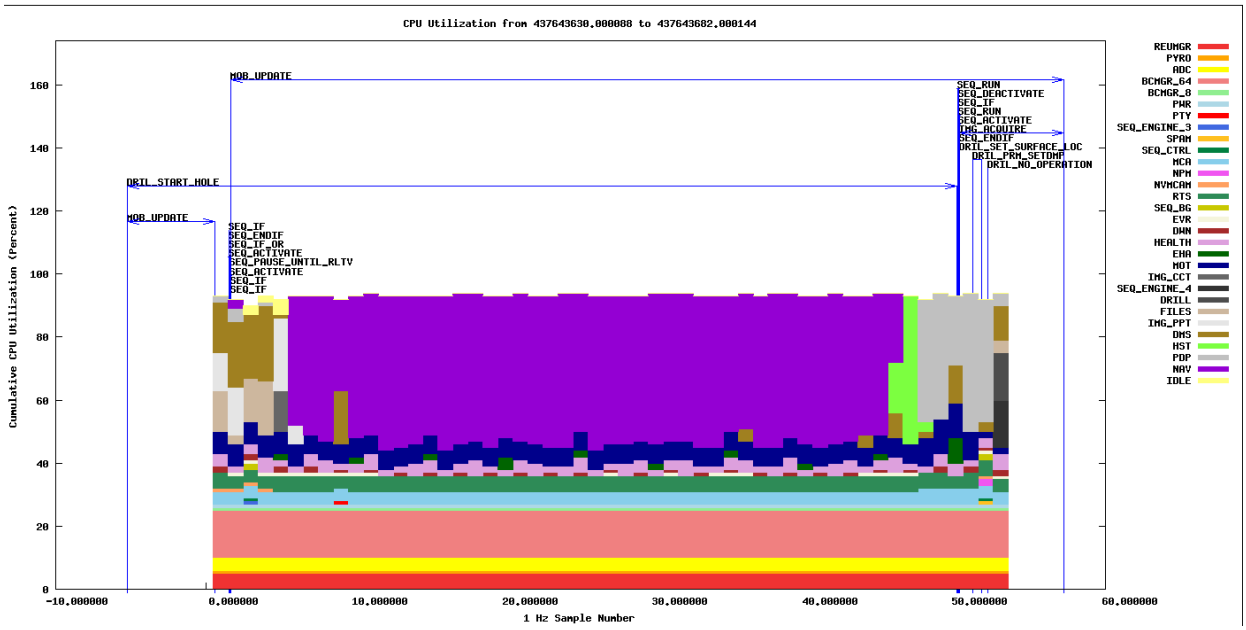

Figure 6. Example of a "Stacked" output plot during nominal Visual Odometry operations. The NAV (purple) processing begins 5 seconds into the command execution.

operations. In this case, VO Monitoring was later used in flight on several sols as an additional safety factor during some drilling operations.

7. Current and Future Uses

CUSP provides the visualizations used to assess CPU usage performance on the Curiosity and Perseverance Mars Rovers by the Mobility, Surface Sampling System (SSS, including Arm), and Flight Software teams. Every Rover Planner "backbone" sequence is bookended by commands that begin and end the collection of CPU usage data with statistics, typically using the 1 Hz sampling rate. The duration of these sequences varies widely, from a few minutes to a few hours, depending on the activities of the day. CPU usage data are downloaded every sol on which a Rover Planner backbone executes. Table 5 summarizes information about the collection of CPU usage information on Curiosity and Perseverance.

MSL FSW, Mobility, and SSS operations teams are relying on this tool to understand performance of the MSL FSW both on the Curiosity rover itself, and during Vehicle System Testbed (VSTB) operations. MSL Mobility Rover Planners have used it to understand Curiosity's driving behavior on Mars and in the testbed, and the Arm Rover Planners have used it to assess daily operations and the behavior of MSL FSW during Drill parallel sequence operations in High Tilt testbed scenarios.

CUSP can be beneficial to any mission or activity that can provide time-stamped logs of CPU utilization numbers and interesting events. Both Realtime operating systems and non-realtime systems that rely on multiple interacting sub-processes can benefit from CUSP's ability to automatically generate easily-understood plots and tables from a raw dump of CPU utilization data. Some other missions or frameworks that could benefit include the FSW CORE project, the Fast Traverse project, and the Europa Habitability Mission.

8. Conclusion

The CUSP tool plots CPU Utilizations for multiple parallel tasks that humans can use to understand where system CPU time is being spent. It can be run in a completely automated way and still produce plots that are helpful for humans to understand and explain system behavior. Ordering the presentation of tasks by their CPU Utilization variance is a novel and helpful way to understand the behavior of parallel tasks.

CUSP is actively being used in Mars Science Laboratory and Mars 2020 operations and FSW development, and also has broader applicability to other CPU trace-generating systems.

9. Acknowledgements

The MSL Resource Tracking Service (RTS) flight software (FSW) provided the raw data that led to the development of CUSP. MSL RTS FSW was written collaboratively by multiple MSL FSW team members including Ben Cichy, Mark Maimone, Udo Wehmeier, and Muh-Wang Yang.

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with NASA.

References

[1] E. Benowitz and M. Maimone. Patching flight software on Mars. In *Workshop on Spacecraft Flight Software*, Laurel, MD, USA, October 2015. https://www-robotics.jpl.nasa.gov/publications/Mark_Maimone/FSW2015_Benowitz_Maimone_4.pdf.

[2] Mikael Johansson and Martin Saegebrecht. Cpu workload analysis in real time operating systems. June 2007.

[3] Rajeev Joshi. Managing data for curiosity, fun and profit. In *Workshop on Spacecraft Flight Software*, 2013.

http://flightsoftware.jhuapl.edu/fsw13.html.

[4] Hisham Muhammad. htop - an interactive process viewer for linux. http://htop.sourceforge.net/, 2014.

[5] Arturo Rankin, Mark Maimone, Jeffrey Biesiadecki, Nikunj Patel, Dan Levine, and Olivier Toupet. Driving curiosity: Mars rover mobility trends during the first seven years. *Journal of Field Robotics*, January 2021.

[6] Kathryn Anne Weiss. The mars science laboratory flight software – a platform for science and mobility. In *Workshop on Spacecraft Flight Software*, 2012. http://flightsoftware.jhuapl.edu/fsw12.html.

BIOGRAPHY

Mark Maimone *is a Robotic Systems Engineer in the Robotic Mobility group at the Jet Propulsion Laboratory. Mark designed and implemented the GESTALT self-driving surface navigation Flight Software for MER and MSL missions; during MSL operations served as Deputy Lead Rover Planner, Lead Mobility Rover Planner and Flight Software Lead; developed downlink automation tools for MER and MSL; and is now working on Mars 2020 as FSW developer and Deputy Lead Rover Planner. He holds a Ph.D. in Computer Science from Carnegie Mellon University.*