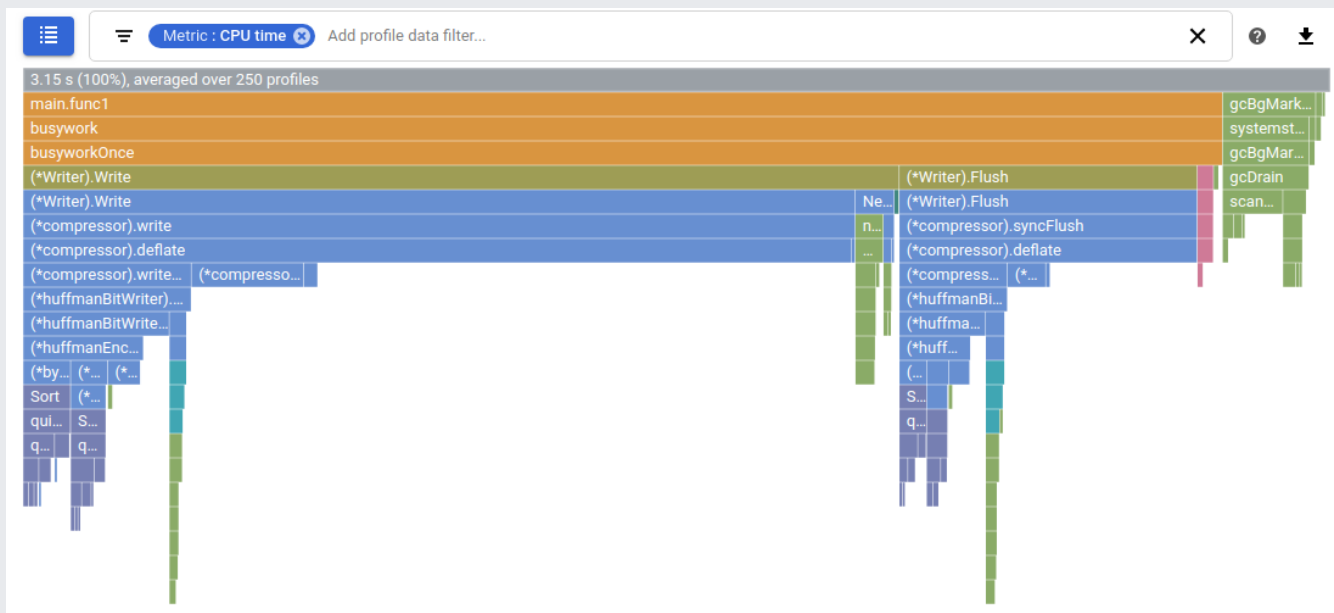


When you use the **Focus** filter, you select a single *function*, and the flame graph displays the code paths that flow into, and out of, that specific function. A focused graph lets you perform two common tasks:

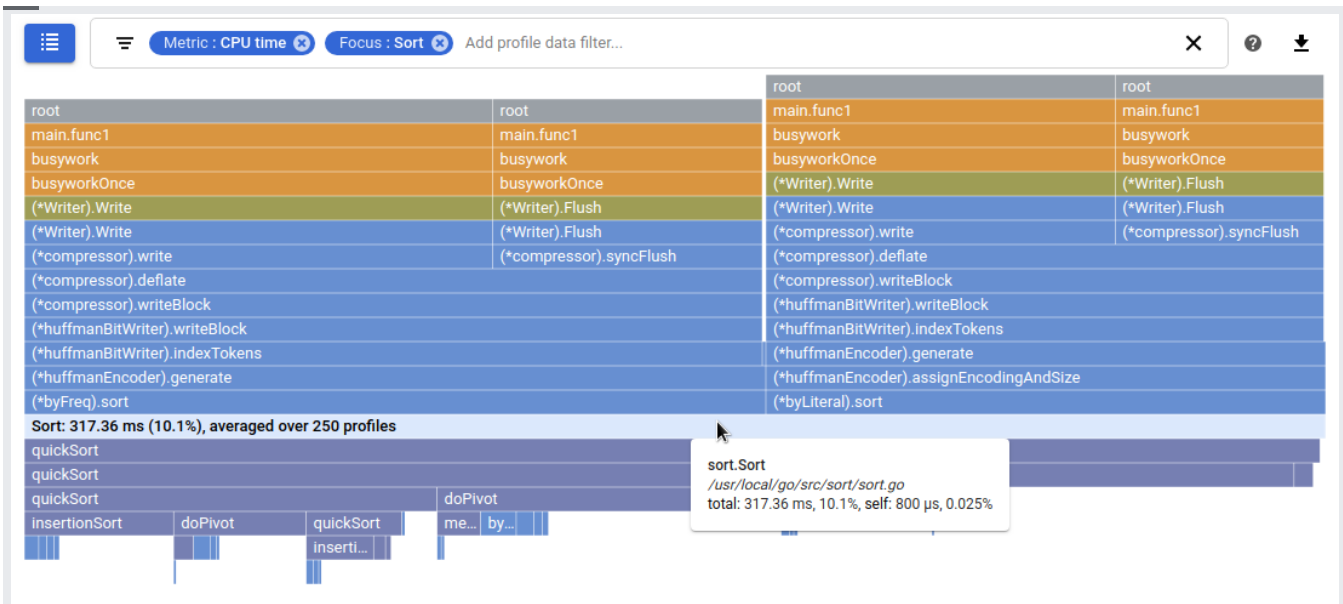
1. Analyzing the aggregate resource consumption of a given function that is called from multiple places.
2. Analyzing the proportion of time spent in a function for different callers of the function.

For example, how do you analyze the resource consumption around the `Sort` function by using the standard flame graph?



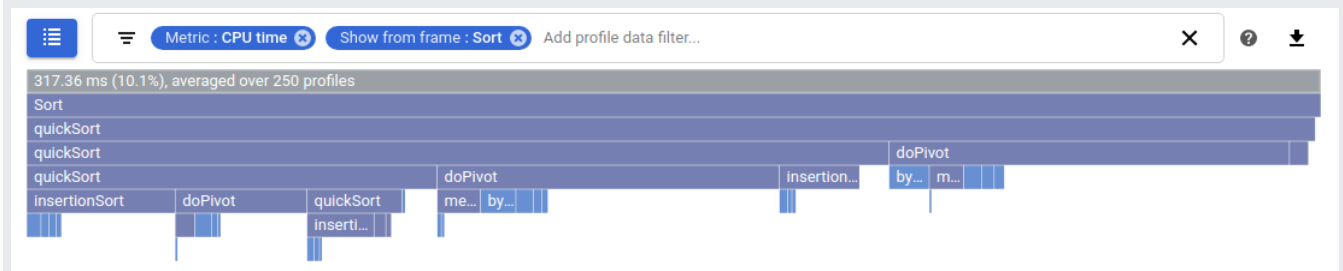
In the next section, we focus the graph on `Sort` and answer this question. The flame graphs on this page were constructed with the **Color mode** and **Compare to** set to the default values of **Name** and **None** respectively.

The graph built by the **Focus** filter effectively creates two flame graphs for the specified function and joins them together:

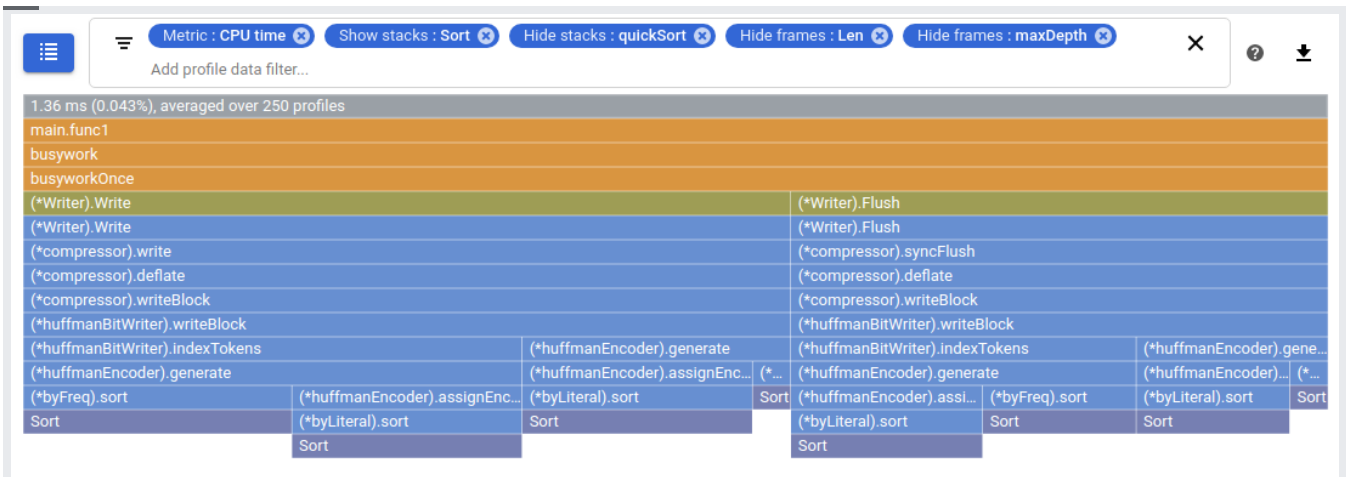


In the preceding graph, the frame corresponding to the Sort function is full width and highlighted. The frame text includes the function name, a percentage, and the number of profiles used for the analysis. In this case, the metrics indicate that the Sort function, in aggregate, consumed 8.85% of the CPU time.

The bottom half of the preceding graph treats the function Sort as the starting point of a standard flame graph and shows all of its callees. You can create this part with the standard flame graph using the **Show from frame** filter:



The top half of the graph shows the callers of Sort with the callees hidden. You can make an approximation of the top half using a series of filters. Start by adding a **Show stacks** filter for Sort. Next, for each function called by Sort, add either a **Hide stacks** or **Hide frames**. In this situation, you would add a **Hide stacks** for quickSort to eliminate this function and its children, and then add **Hide frames** for Len and maxDepth:



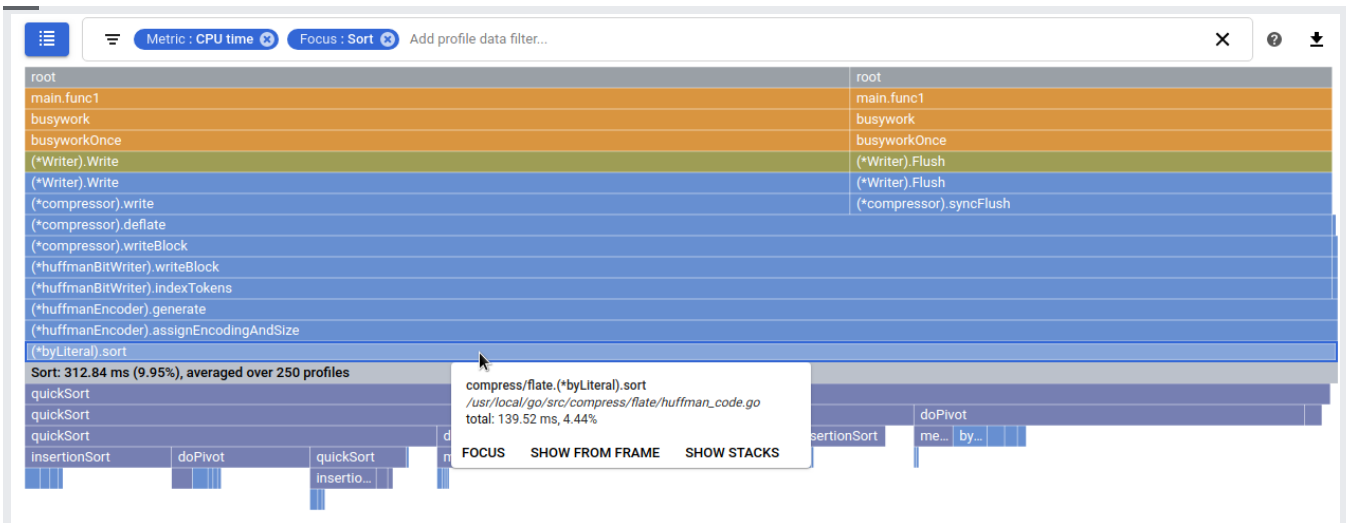
Using these filters, the approximation of the top half of the focused graph shows that the `Sort` function is reached through different call stacks. However, the metrics aren't aggregated so the graph doesn't illustrate overall metric consumption by `Sort`.

The focused graph is a little different than a graph that just combines the two approximations:

- There is a single frame for the focus function `Sort`.
- The focus function frame is highlighted, is full-width frame, and displays metrics that are the aggregation of all call stacks.
- There are multiple call stacks, each beginning with a root frame, so you can view the entire call stack.

If you select a frame in a focused graph, then the flame graph is redrawn with that frame's call stack displayed in more detail. If the frame is reached through multiple call stacks, each of those call stacks is displayed. Call stacks that don't include the frame are hidden from view. To restore the graph to its original state, select the frame that corresponds to the focus function.

In the previous example, `Sort` is called by `(*byFreq).sort` and by `(*byLiteral).sort`. To view the call stack for `(*byLiteral).sort` in more detail, select that frame. You can select another frame and further refine the call stacks being displayed:



To restore a focused flame graph to its original state, select the frame that displays the value of the **Focus** filter. In this case, select the gray frame with the label **Sort**. Note that to restore a standard flame graph to its original state you select the root frame.

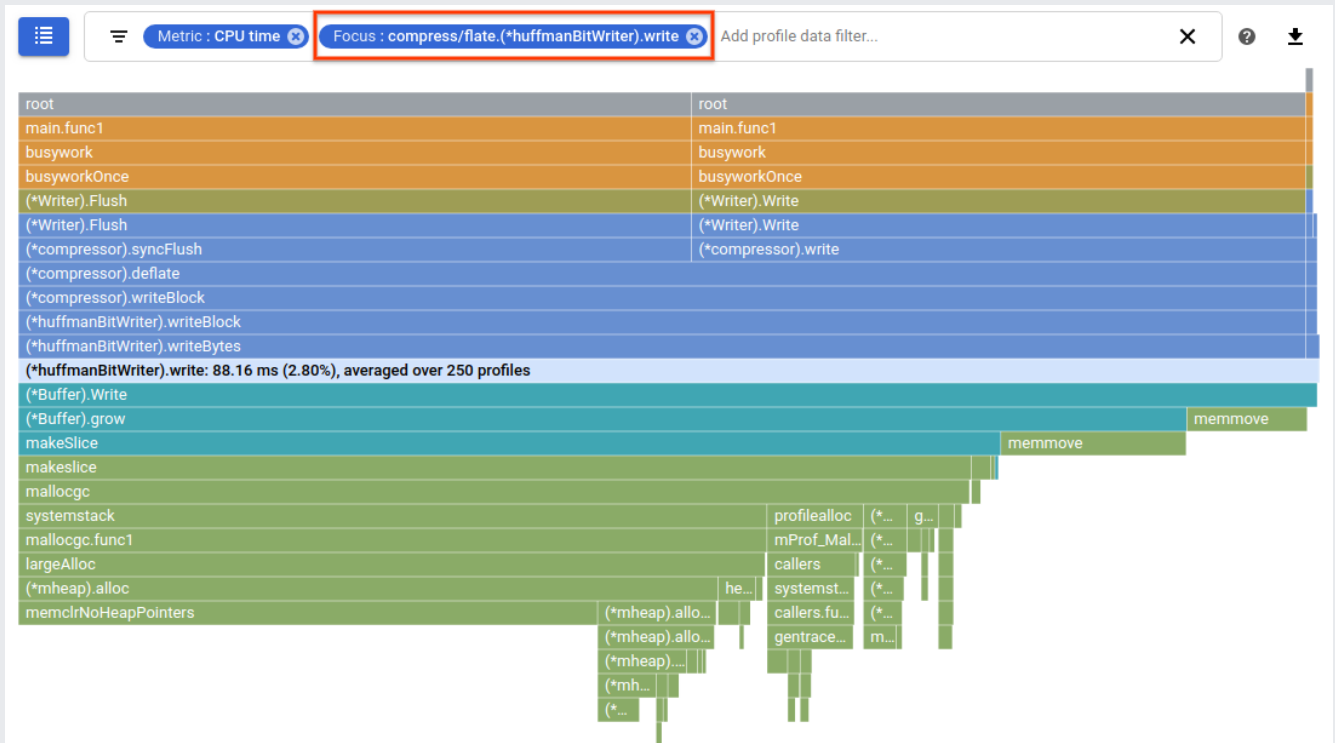
To analyze a focused flame graph, you use the same controls and filters that you use to analyze a standard flame graph. However, there are differences in how the graphs interact with the pointer:


- If your pointer hovers on a frame, the tooltip displays metric data. For a standard flame graph, total metric data for the frame is shown. For a focused flame graph, aggregate metric data for the function is shown.
- If you select a frame, the flame graph is redrawn with that frame displayed full width. To restore a standard flame graph to its original form, you must select the top frame. To restore a focused flame graph to its original form, you must select the frame that displays the value of the focus filter.

For information on the focused graph when you are comparing profiles, see [Focusing a comparison](/profiler/docs/comparing-profiles#with-focus) (/profiler/docs/comparing-profiles#with-focus).

There are different methods you can use to set a focus filter but they result in the same graph.

Place your pointer on the frame of interest, and then click **Focus** in the frame tooltip. The focus function is extracted from the frame. In this example the flame graph, which is expanded around the function `(*huffmanBitWriter).write`, displays three different call stacks:



To focus the flame graph on a specific function, click **List** , and then select a row from the **Select focus function** table:

Select focus function					
Name	↓ Self	%	Total	%	Count
(*compressor).deflate	971 ms	62%	1.46 s	93%	2
(*compressor).findMatch	218.96 ms	14.0%	219.04 ms	14.0%	2
(*huffmanEncoder).bitCounts	41 ms	2.62%	51.8 ms	3.31%	3
(*huffmanBitWriter).indexTokens	38.08 ms	2.43%	202.52 ms	12.9%	2
memmove	34.84 ms	2.22%	34.84 ms	2.22%	11
doPivot	32.16 ms	2.05%	73.36 ms	4.68%	15
memclrNoHeapPointers	25.24 ms	1.61%	25.24 ms	1.61%	16

If the function you selected can be called through different call stacks, each call stack is shown in the flame graph.

You can sort the table rows in ascending \uparrow or descending \downarrow order by selecting a table header element. Each row in the table displays a function name and statistics related to the function's execution. This table shows that the `(*compressor).deflate` function requires 1.46 s to execute, with 971 ms spent in the function itself and the remainder of the time spent in its call stack. One percentage column reports that 62% of the total execution time is spent in the function `(*compressor).deflate`. Another column reports that 93% of the time, `(*compressor).deflate` or a function in its call stack, is executing. Lastly, the count column reports that there are 2 sequences that invoke the function `(*compressor).deflate`.

When you are comparing profiles, the content of the focus list is different. For more information, go to [Focusing a comparison](/profiler/docs/comparing-profiles#with-focus) (/profiler/docs/comparing-profiles#with-focus).

Click the gray text **Add profile data filter** in the filter bar, and then enter `Focus:` and a string that identifies the function to focus on. You can use a substring, including package prefixes, or the full name. When you supply an ambiguous string, the function that is the best match to the string is selected.

If you prefer, you can click **Filters**, select **Focus**, and then enter the identifying string.

If the function you selected can be called through different call stacks, each call stack is shown in the flame graph.

To remove the focus filter, click **Close** **×** on the filter.

- For information on comparing profiles collected by different deployments of your service, see [Comparing profiles](/profiler/docs/comparing-profiles) (/profiler/docs/comparing-profiles).
- To learn how to download your profile data, see [Downloading profiles](/profiler/docs/downloading-profiles) (/profiler/docs/downloading-profiles).
- For information on using the Profiler agent to collect profiling data for your services, see:
 - [Profiling Go applications](/profiler/docs/profiling-go) (/profiler/docs/profiling-go)
 - [Profiling Java applications](/profiler/docs/profiling-java) (/profiler/docs/profiling-java)
 - [Profiling Node.js applications](/profiler/docs/profiling-nodejs) (/profiler/docs/profiling-nodejs)
 - [Profiling Python applications](/profiler/docs/profiling-python) (/profiler/docs/profiling-python)
 - [Profiling applications running outside Google Cloud](/profiler/docs/profiling-external) (/profiler/docs/profiling-external)