VisPerf: visualize and compare performance of stream processing applications executed on CPU

Claudio Scheer* Dalvan Griebler† Isabel Harb Manssour‡

Pontifical Catholic University of Rio Grande do Sul - PUCRS
Brazil

ABSTRACT

This is my abstract.

Index Terms: Human-centered computing—Visualization—Visualization techniques—Treemaps; Human-centered computing—Visualization—Visualization design and evaluation methods

1 INTRODUCTION

This is the introduction.

2 PERF

Perf is a profiling tool included in the Linux kernel. It can be used to capture CPU performance counters, tracepoints, kprobes[1], and uprobes[2] [2]. Perf also supports software events, such as page misses. In turn, tracepoints are placed in the source code to collect timestamps and stack traces and performance counters are hardware registers that count events of the CPU, such as instructions executed and cache-misses. Perf has low overhead, as it is integrated into the kernel. Moreover, Perf supports counters in different architectures. Therefore, it can be used to profile applications in different Intel architectures or even in AMD CPUs.

There are several subcommands available in Perf. Below we describe some of the subcommands:

- `perf stat`: gather performance counter statistics from a command;
- `perf record`: run a command and record its profile for later analysis. By default, profile is saved in a file named `perf.data`;
- `perf report`: read `perf.data` and display the profile recorded in this file;
- `perf script`: read `perf.data` and display the trace recorded in this file;
- `perf top`: display performance counter profile in real time;
- `perf list`: list events available to be captured in the current architecture;

*e-mail: claudio.scheer@edu.pucrs.br
†e-mail: dalvan.griebler@edu.pucrs.br
‡e-mail: isabel.manssour@pucrs.br

[1] `https://www.kernel.org/doc/html/latest/trace/kprobes.html`

[2] `https://www.kernel.org/doc/html/latest/trace/uprobetracer.html`

`perf list` shows events from several sources, such as hardware and software. Hardware events are provided by the processor and its PMUs (Performance Monitoring Unit), which may vary among different companies. Next, we discuss the counters captured in our experiments [1].

- **cpu-cycles**: number of fetch-decode-execute cycles the CPU executed;
- **instructions**: number of instructions the CPU executed;
- **cache-misses**: number of cache-misses;
- **cache-references**: number of references found in cache;
- **L1-dcache-load-misses**: number of misses when loading data from L1 cache;
- **L1-dcache-loads**: number of data loads from L1 cache;
- **L1-dcache-stores**: number of data stores made in L1 cache;
- **L1-icache-load-misses**: number of misses when loading instructions from L1 cache;
- **L1-icache-loads**: number of instructions load from L1 cache;
- **L1-icache-stores**: number of instructions store made in L1 cache;
- **LLC-loads**: number of references loaded from LLC (Last Level Cache, usually L3);
- **LLC-load-misses**: number of references missed when loading from LLC;
- **LLC-stores**: number of references store made loading in LLC;
- **mem-stores**: number of stores made in the main memory;
- **mem-loads**: number of loads made from the main memory;
- **branch-misses**: number of mispredicted speculative branches;
- **branch-instructions**: number of correct speculative branches;
- **bus-cycles**: number of CPU cycles needed to fetch or write data to the main memory, for example;

For this paper, we profiled the person recognition application. The main goal of this application is to recognize is there is a person in a specific image, using OpenCV. To characterize a stream processing application, it is, we cannot predict when data will stop entering the application, we used a video as input.

Towards exploiting parallelism, we use the FastFlow library and compared two mapping policies. In summary, the mapping policy defines in which core threads launched by the application will be placed. We profiled the person recognition application using the default mapping policy os the Linux operating system and the mapping policy proposed by FastFlow.

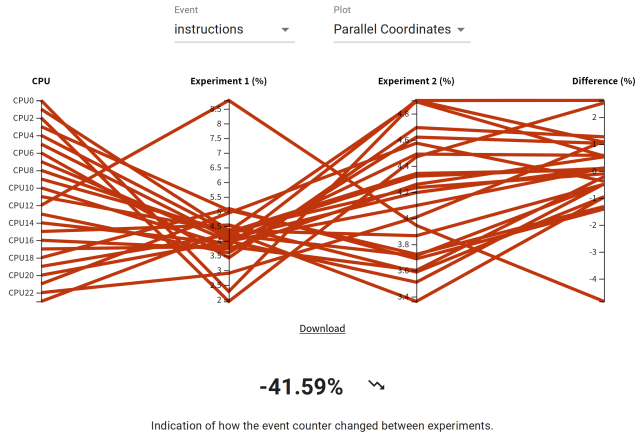

Figure 1: VisPerf section “Comparing experiments”.

3 VisPerf

Since Perf can output a lot of data, we propose VisPerf to easily visualize and compare the data captured. The VisPerf pipeline can be seen as three stages: capture, pre-processing and visualization.

In the capture stage, the person recognition stream application is executed using the mapping policies from operating system and FastFlow, and profiled with Perf. We configured Perf to sample 997 captures each second the application is running. The raw data from Perf is converted to a CSV file for posterior analysis.

In the second stage, the CSV files are processed and merged. In summary, the processing phase groups captures made inside the same second, extract the three top functions in the stack executing on the CPU, and also setting the label to the threads. This stage generates a JSON file that is upload to VisPerf dashboard for human analysis. The JSON file contains the structured data of each experiment and also the details of the hardware where experiments where executed, enabling fast data reading on the VisPerf dashboard. Moreover, generating a single JSON file with all experiment makes it simple to share data for analysis between researchers.

After these two stages finished, VisPerf can be to analyze data captured. In Section 3.1, we discuss the visualizations and interactions we propose for VisPerf.

3.1 Visualizations and Interactions

The first step inside VisPerf is to upload the JSON file generated in the second stage. VisPerf reads this file and generate the plots. We used D3[3] library to create the visualizations, since it is very customizable and with many features to allow users to interact with the plots. Since the JSON file file may contains many experiments, we allow the user to select which experiments they want to compare. The user can select up to two experiments.

VisPerf visualization is divided into three sections. The first section, showed in Figure **??**, is named “Comparing experiments”. This section compares specific events captured while executing the two experiments selected. As shown in Figure 1, the user can select the event to be compared and the plot. For this sections, there are two plots available: the parallel coordinates and another plot with a grid representing the CPUs available in the processor where experiments were executed.

The first section in VisPerf also shows a summary of how the event selected change between the experiments. As shows in Figure 1, the second experiment needed 41.59% less instructions to finish the execution. Another important metric that this plot can shows is

[3] `https://d3js.org`

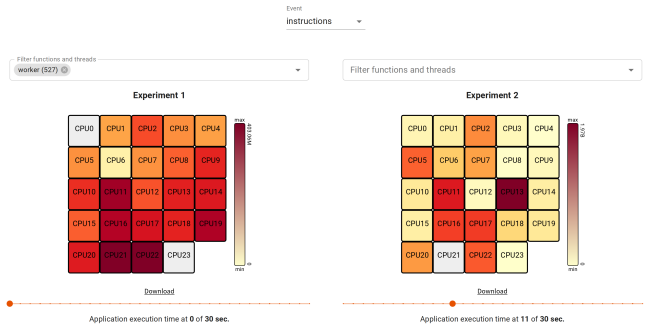

Figure 2: VisPerf section “Comparing experiments over execution time”.

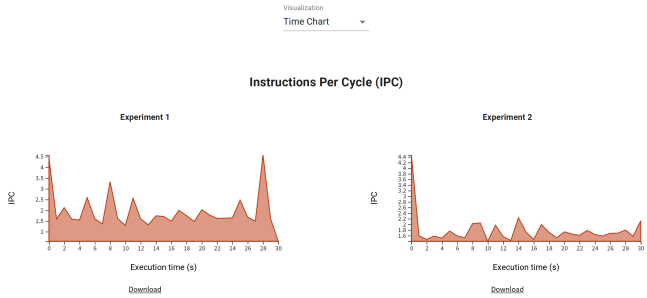

Figure 3: VisPerf section “Performance evaluation metrics”.

that the second experiment had a better balance between the CPUs. CPU 12, for example, executed ≈ 9% of the instructions, while in experiment two the most CPUs executed ≈ 5% of the instructions. The last columns in the parallel coordinates plots shows the absolute difference between the CPUs.

The parallel coordinate plots seems to be a little messy because of the high number of lines. However, when hovering a specific line, this line is highlighted and the opacity of other lines diminished. Another feature that is present in all plots is the download button. Here the plot is save in SVG format, that can be converted to any other format by the user.

The second section, named “Comparing experiments over execution time” and shown in Figure 3, allows to explore the performance of specific functions or threads executed in the program. Since Perf only output the thread id, we modified the application source code to output which was the function being executed in that specific threads. Therefore, thread ids that are not present in the program output are named as “Other threads”. In this section, the user can see the performance of the PMUs at specific times of the application execution. Navigating in the experiment one, shown in Figure 3, we can see that CPUs 0 and 23 were not used. While in experiment two, all CPUs are used, and in some points some threads migrated of CPU while running the person recognition application. This behavior is correct because experiment one is using a fixed mapping, proposed by FastFlow, and experiment two the default mapping of the operating system.

Last section of VisPerf has metrics that use the events captured by Perf. For now, we only have one metrics: IPC (Instructions Per Cycle). This metric indicates the average number of instructions that were executed at each CPU cycle.

4 Conclusion

References

[1] collectd Wiki. *Plugin:Intel PMU - collectd Wiki*, June 2021.
[2] Linux Kernel Organization, Inc. *Perf Wiki*, May 2021.