Holoscan Flow Benchmarking

White Paper

Version 1.0 | November 2023

Document History

Version 1.0

Version	Date	Authors	Description of Change
1.0	November 27, 2023	P. Gambrill M. Toloui	Initial release

Table of Contents

Overview

Developing and optimizing the performance of a time-critical application is a challenging task. One needs to run an application with different configurations and evaluate the performance to figure out the desired production environment of an application. Holoscan is a real-time AI sensor-processing platform that enables developers to build real-time AI applications with intuitive interfaces. However, measuring the performance of a Holoscan application can require significant efforts by the application developers. This article introduces Holoscan flow benchmarking on HoloHub, which aims to make Holoscan applications performance measurements and analysis easy and scalable. Holoscan flow benchmarking can help evaluate different configurations before deploying a Holoscan application to production.

A key performance indicator (KPI) for real-time sensor processing applications is the end-to-end latency. There are many possible metrics on the end-to-end latency performance, including average end-to-end latency and maximum (worst-case) end-to-end latency. For a developer, these metrics indicate how well their application is performing in terms of responsiveness and efficiency. Holoscan SDK data flow tracking allows for precise measurement of the end-to-end latency of Holoscan applications. Holoscan flow benchmarking in Holohub leverages the data flow tracking of the Holoscan SDK.

Introduction to Holoscan Flow Benchmarking

Holoscan flow benchmarking allows developers to automate the benchmarking process and generate the necessary log files, which are analyzed later. This article demonstrates how to use Holoscan flow benchmarking to evaluate different configurations, applications and schedulers.

There are four steps to perform Holoscan flow benchmarking, which are summarized here. The following is a high-level overview:

1. **Patching an application:** Patch a Holoscan application to turn on data flow tracking.
2. **Building an application:** Compile the patched application with necessary header files.

3. **Running the benchmark:** Run the `benchmark.py` script with necessary parameters to evaluate the performance of the application in different configurations. This generates a number of log files that capture the result of the performance measurements in a raw format.
4. **Analyzing the result:** Retrieve the result of the benchmarking process using the generated log files and the `analyze.py` script. You can also export the results in a CSV file or generate graphs to visualize certain results.

The following sections will demonstrate some of the capabilities of Holoscan flow benchmarking.

Comparing performance for multiple instances of an application

In many cases, users may want to run multiple Holoscan applications in parallel on their systems. They want to know the performance implications of a particular configuration before developing a full-fledged system with more than one Holoscan application. Through Holoscan flow benchmarking, they can easily evaluate the performance of multiple instances of an application in parallel to compare the behavior of specific and curated use-cases. They can learn how many parallel instances of an application, representative of their own application, can safely run until the configuration cannot meet their expected end-to-end latency guarantees.

Suppose you want to evaluate the performance of running multiple instances of the endoscopy tool tracking application in parallel to understand end-to-end latency performance. After completing the first two steps specified in the **Introduction to Holoscan Flow** workflow, you can run the following command to evaluate how up to 5 concurrent instances of the endoscopy tool tracking application will perform.

```
$ python tutorials/benchmarking/benchmark.py -a endoscopy_tool_tracking --sched greedy -i <Number of Instances> -r 10 -m 1000 -d endoscopy_<Number of Instance>
```

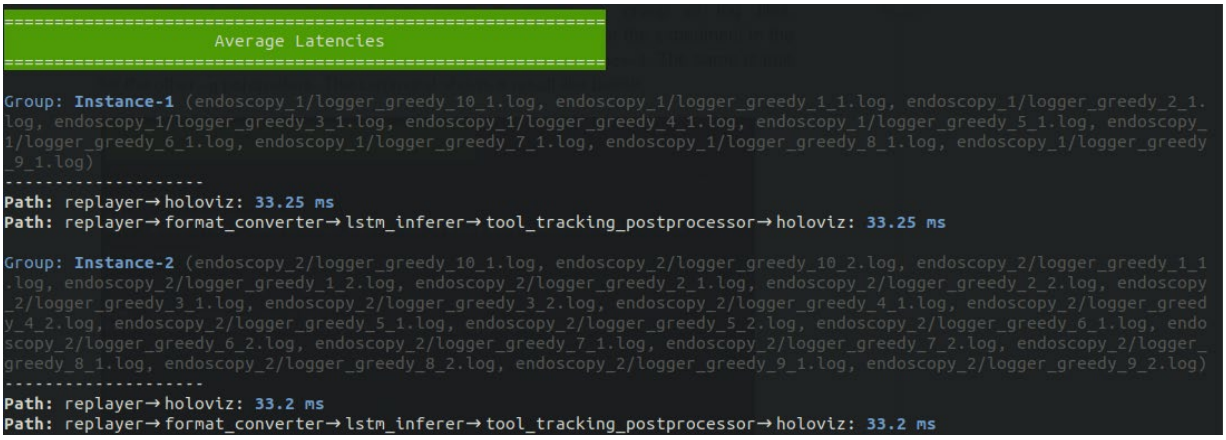
`Number of Instances` needs to be set to measure the performance of multiple concurrent instances of the endoscopy tool tracking applications. The above command tests the application against 1000 data frames (`-m 1000`) and repeats every experiment 10 times (`-r 10`) for the greedy scheduler (`--sched greedy`). The log files generated by the benchmarking process are stored in the `endoscopy_<Number of Instance>` directory, where `Number of Instance` varies from 1 to 5. The format of the filename for the data flow tracking log files is `logger_<scheduler>_<run_number>_<instance-id>.log`.

Retrieving the Result

After the benchmarking process is complete, you can analyze the result using the `analyze.py` script. If you want to know the average end-to-end latencies of the above experiment, the following command could be used:

```
$ python tutorials/holoscan_flow_benchmarking/analyze.py --avg -g
endoscopy_1/logger* Instance-1 -g endoscopy_2/logger* Instance-2 -g
endoscopy_3/logger* Instance-3 -g endoscopy_4/logger* Instance-4 -g
endoscopy_5/logger* Instance-5
```

In the above command, `--avg` is used to show the average end-to-end latency of each experiment. Every `-g` parameter is used to specify a group of log files. `endoscopy_1/logger*` specifies the data flow tracking log files for the experiment in the `endoscopy_1` directory. It is followed by a group name of `Instance-1`. The same is true for the other `-g` parameters. Executing the command shows a result similar to following:



Plotting the Data

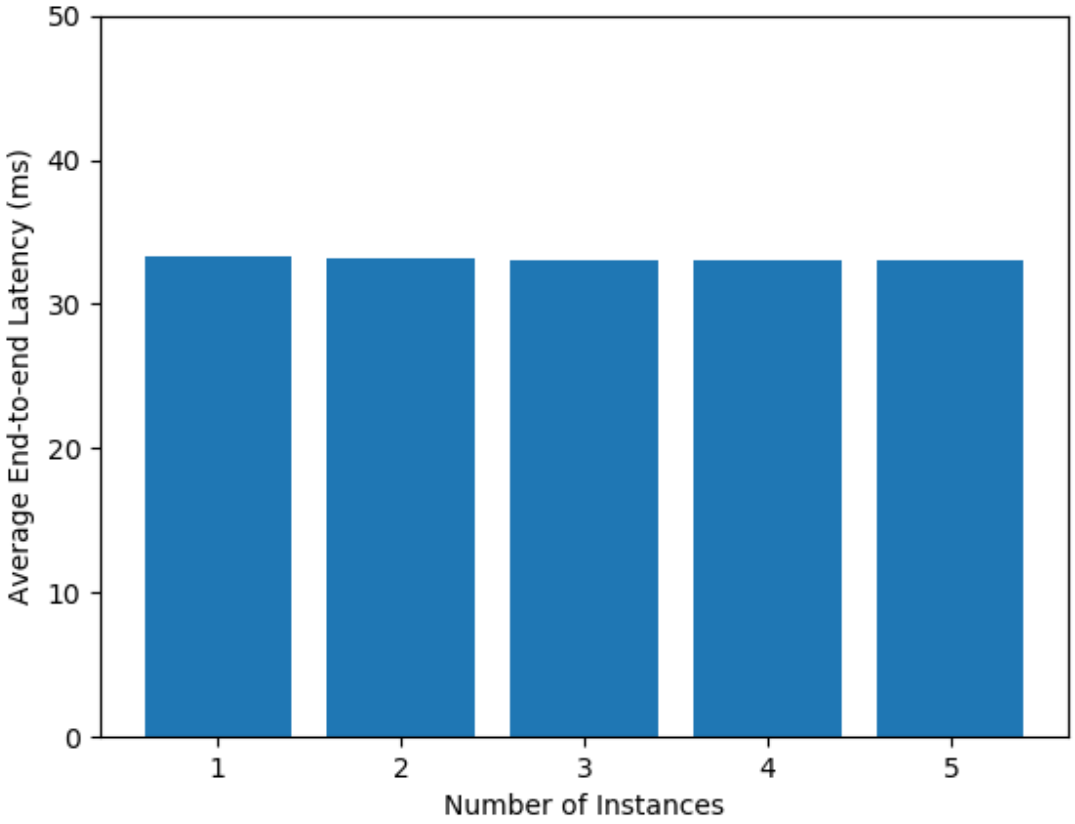
In addition to getting the analysis results, it is important to plot the result in a graph to better understand the trends in an evaluation result. The Holoscan flow benchmarking provides options to export the results in a CSV file to plot the data with your favorite plotting tool later on. Appending the `analyze.py` command with the `--save-csv` parameter enables saving the results in an `avg_values.csv` file in CSV format:

```
$ python tutorials/holoscan_flow_benchmarking/analyze.py --avg -g
endoscopy_1/logger* Instance-1 -g endoscopy_2/logger* Instance-2 -g
endoscopy_3/logger* Instance-3 -g endoscopy_4/logger* Instance-4 -g
endoscopy_5/logger* Instance-5 --save-csv
```

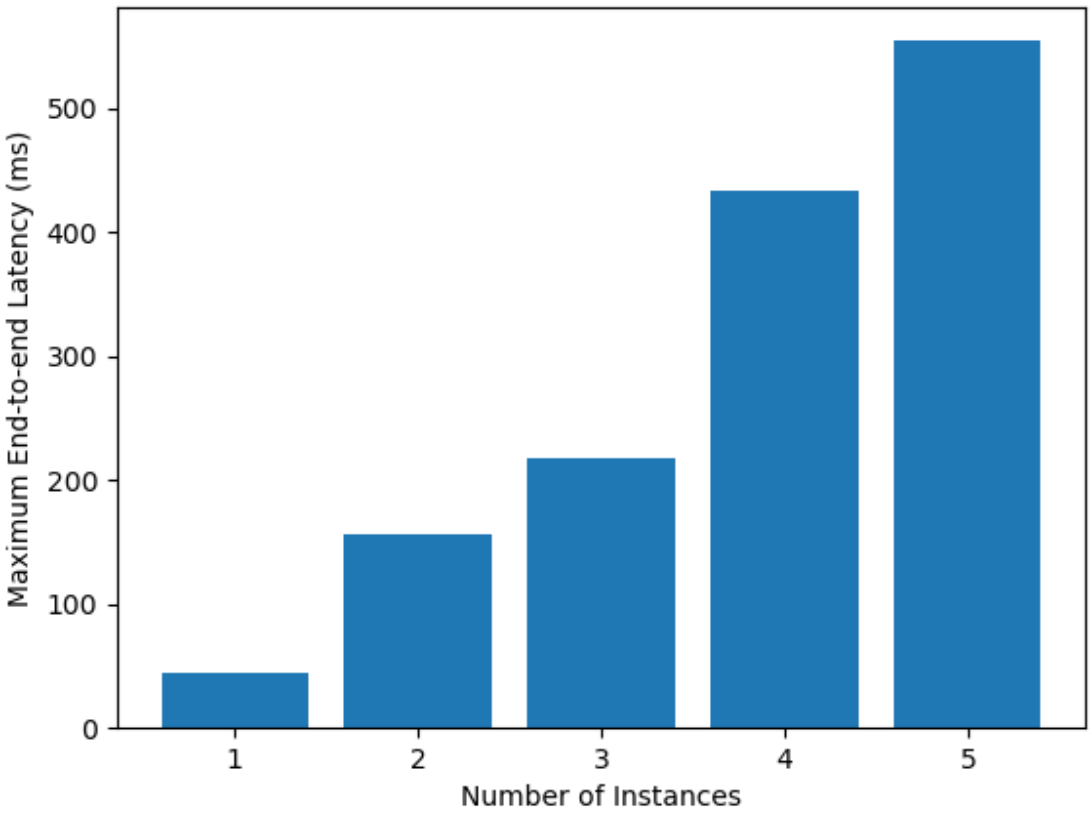
Then, the average values could be plotted with your preferred plotting tools. For example, the `matplotlib` Python library could be used to plot the average values in a bar-graph with the following code:

```
import matplotlib.pyplot as plt

with open('avg_values.csv', 'r') as f:
    lines = f.readlines()
    avg_values = [float(x) for x in lines[0].strip().split(',') if x]

plt.bar(range(1, 6), avg_values)
plt.xlabel('Number of Instances')
plt.ylabel('Average End-to-end Latency (ms)')
plt.savefig('avg_bar_graph.png')
```

The graph looks like the following:



Similar results can be obtained for the maximum end-to-end latencies as well if the above `analyze.py` command is supplied with the `--max` parameter. The bar-graph with the maximum end-to-end latency would look like the following graph, as the worst-case latency is affected by interferences from multiple instances:



The application developers can decide for their own applications the acceptable maximum or average end-to-end latency and, as a result, determine the following:

- How many parallel instances of an application they want to run on their system
- Whether they need to further optimize the pipeline
- Whether they need more compute resources

Including GPU Utilization in the Result

Understanding the impact of running multiple instances of an application on GPU utilization is crucial for optimizing performance. Holoscan flow benchmarking can also help measure the GPU utilization for every experiment. By appending `-u` to the above `benchmark.py` command, the GPU utilization is monitored and logged in a CSV file named `gpu_utilization_<scheduler>_<run_number>.csv`. For example, the final command for GPU utilization monitoring is as follows:

```
$ python tutorials/benchmarking/benchmark.py -a endoscopy_tool_tracking --sched greedy -i
<Number of Instances> -r 10 -m 1000 -d endoscopy_<Number of Instance> -u
```

In the analysis phase, the average GPU utilization result can be saved in a CSV file (`avg_gpu_utilization_values.csv`) using the `-u --save-csv` parameters:

```
$ python tutorials/holoscan_flow_benchmarking/analyze.py -g endoscopy_1/logger* Instance-
1 -u endoscopy_1/gpu* Instance-1-GPUUtil -u endoscopy_2/gpu* Instance-2-GPUUtil -u
endoscopy_3/gpu* Instance-3-GPUUtil -u endoscopy_4/gpu* Instance-4-GPUUtil -u
endoscopy_5/gpu* Instance-5-GPUUtil --save-csv
```

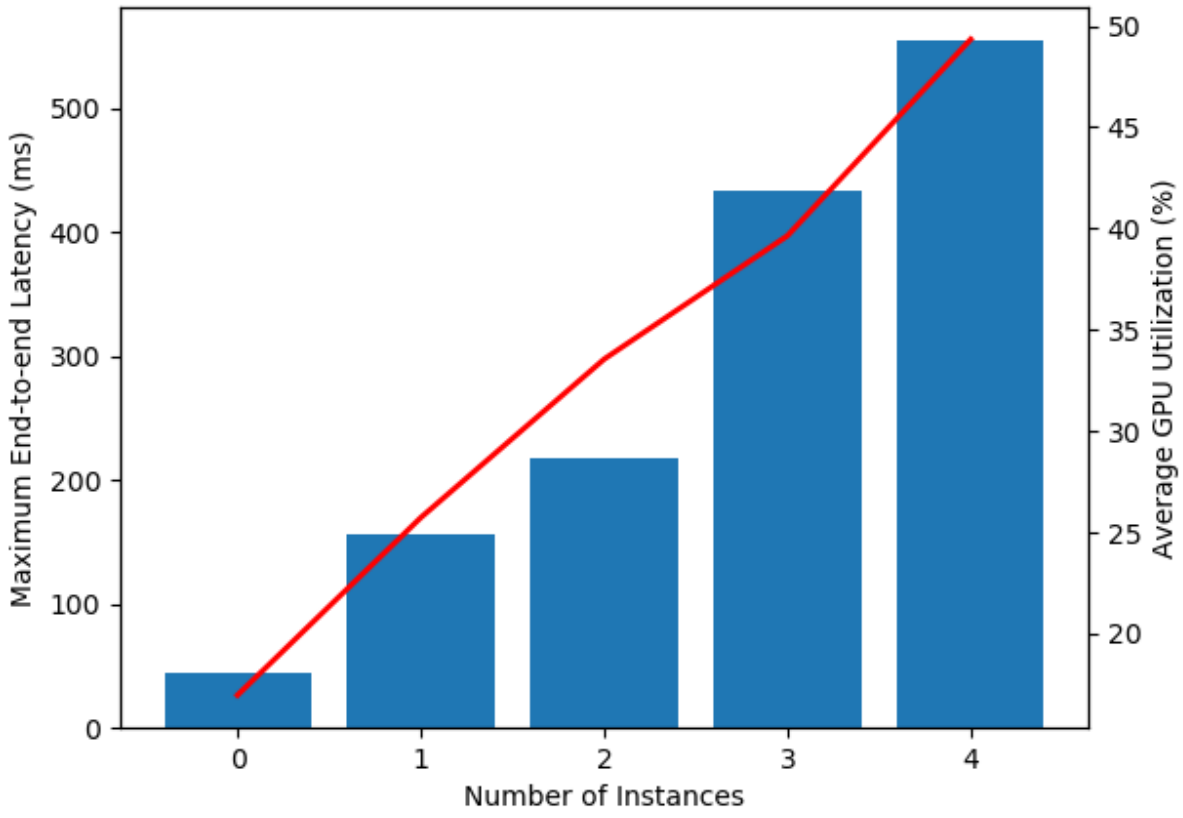
Each `-u` parameter specifies the group of GPU utilization log files (`endoscopy_1/gpu*`), with a last parameter that specifies the group name (`Instance-1-GPUUtil`). `-g endoscopy_1/logger* Instance-1` to satisfy the requirement of a mandatory `-g` parameter.

Suppose you want to plot the average GPU utilization, as well in the above bar-graph of maximum values. You can achieve that with the following code in Python:

```
import matplotlib.pyplot as plt

with open('max_values.csv', 'r') as f:
    lines = f.readlines()
    max_values = [float(x) for x in lines[0].strip().split(',') if x]

with open('avg_gpu_utilization_values.csv', 'r') as f:
    lines = f.readlines()
    avg_gpu_utilizations = [float(x) for x in lines[0].strip().split(',') if x]

fig, ax1 = plt.subplots()
ax1.bar(range(len(max_values)), max_values)
ax1.set_ylabel('Maximum End-to-end Latency (ms)')
ax1.set_xlabel('Number of Instances')
ax2 = ax1.twinx()
```

```
ax2.plot(range(len(avg_gpu_utilizations)), avg_gpu_utilizations, color='r',
linewidth=2.0)
ax2.set_ylabel('Average GPU Utilization (%)')
plt.savefig('max_bar_graph.png')
```

The generated graph looks like the following:

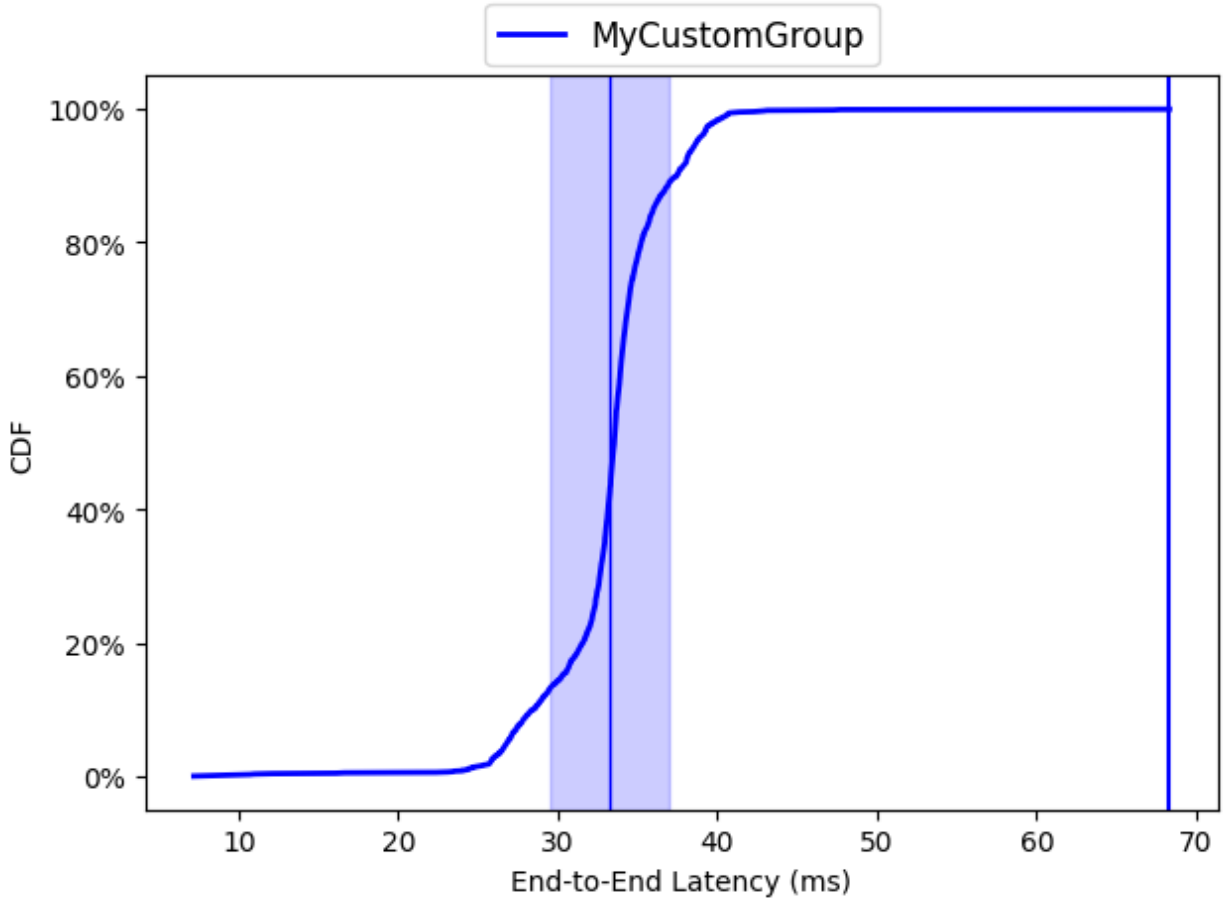


Generating a CDF Curve

Holoscan flow benchmarking can also be used to generate the graph of the Cumulative Distribution Function (CDF) of the observed end-to-end latencies. The CDF curve is a useful tool to understand how much the latencies of all the data frames passing through an application graph are distributed. The following command generates a CDF curve of the end-to-end latencies of the first path for the experiment with one instance:

```
$ python tutorials/holoscan_flow_benchmarking/analyze.py --draw-cdf single_path_cdf.png -g endoscopy_1/logger* MyCustomGroup --no-display-graphs
```

By default, the command also displays the graph in a separate window. The `--no-display-graphs` suppresses this option. The graph is saved as a `single_path_cdf.png` file; an example is shown below:



In addition to plotting the CDF curve of the first path, it is also possible to plot the CDF curves of all the paths of an application graph with the `--draw-cdf-paths` parameter.

Comparing Two Different Applications

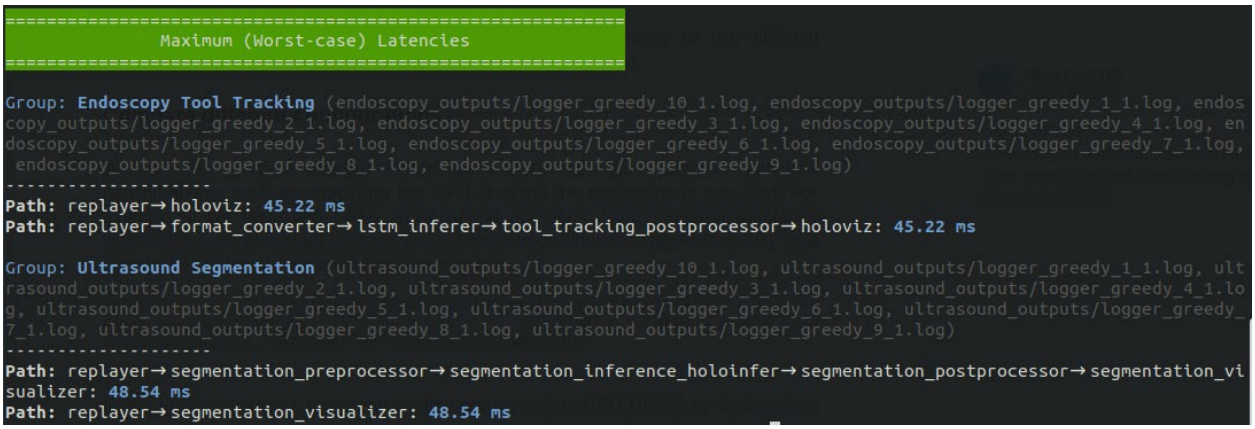
The `benchmark.py` script can be used to evaluate the performance of two different applications. The results can be analyzed and compared using the `analyze.py` script. For example, you can run the following two commands to run two different applications under the same settings:

```
$ python tutorials/holoscan_flow_benchmarking/benchmark.py -a endoscopy_tool_tracking -r 10 -i 1 -m 1000 --sched greedy -d endoscopy_outputs
$ python tutorials/holoscan_flow_benchmarking/benchmark.py -a ultrasound_segmentation -r 10 -i 1 -m 1000 --sched greedy -d ultrasound_outputs
```

The above commands run two experiments and save the corresponding log files in two different directories, `endoscopy_outputs` and `ultrasound_outputs`. We can analyze and compare their maximum end-to-end latencies with the following `analyze.py` command:

```
$ python tutorials/holoscan_flow_benchmarking/analyze.py --max -g endoscopy_outputs/logger* "Endoscopy Tool Tracking" -g ultrasound_outputs/logger* "Ultrasound Segmentation"
```

The result is as follows:



Such a command shows the maximum end-to-end latency of two different applications and helps the developers compare their performances.

Comparing Different Configurations

You can also compare different configurations of an application with Holoscan flow benchmarking, such as specifying the GPU in which the application is executed. For example, if you have two GPUs (RTX A6000 and RTX A4000) both available for visualization, you may want to know the performance of running an application on one of the GPUs only. You can run the following command:

```
$ python tutorials/holoscan_flow_benchmarking/benchmark.py -a endoscopy_tool_tracking -r 10 -i 1 -m 1000 --sched greedy -d endoscopy_outputs_<GPU Name> -g <Corresponding GPU UUID>
```

The above command requires a GPU UUID, which can be looked up using `nvidia-smi -L`, and the `<GPU Name>` can be used to save the outputs in different directories. The `-g <Corresponding GPU UUID>` parameter sets the `CUDA_VISIBLE_DEVICES` environment variable to the corresponding GPU UUID. It is important to note that this option does not override the application-specific GPU settings in the code--for example, via the GPU assignment in Inference operator (`device_map` option).

The results of the above experiment can be compared using the following command:

```
$ python tutorials/holoscan_flow_benchmarking/analyze.py <insert your metrics options> -g endoscopy_outputs_<GPU Name>/logger* "GPU1" -g endoscopy_outputs_<GPU Name>/gpu* "GPU2"
```

Comparing the Performance of Different Schedulers

It is also possible to compare different schedulers using Holoscan flow benchmarking. The following commands can be used to evaluate performance of the endoscopy tool tracking sample application under the Greedy scheduler and the Multithread scheduler:

```
$ python tutorials/holoscan_flow_benchmarking/benchmark.py -a endoscopy_tool_tracking -r
10 -i 1 -m 1000 --sched greedy -d endoscopy_greedy_outputs
$ python tutorials/holoscan_flow_benchmarking/benchmark.py -a endoscopy_tool_tracking -r
10 -i 1 -m
1000 --sched multithread -w 5 -d endoscopy_multithread_outputs
```

Then, the results can be analyzed and compared using the following command:

```
$ python tutorials/holoscan_flow_benchmarking/analyze.py <insert your metrics options> -g
endoscopy_greedy_outputs/logger* "Endoscopy (Greedy)" -g
endoscopy_multithread_outputs/logger* "Endoscopy (Multithread)"
```

Tail and Flatness of the CDF Curve

The `analyze.py` script provides options for calculating the tail and flatness of the CDF curve of the observed latencies using the `--tail` and `--flatness` parameters, respectively.

The tail of the CDF curve is the difference between the 95 percentile and 100 percentile values. It helps to understand how widely distributed the worst-case latencies of an application are. The lesser the value of the CDF curve tail, the better the application is performing in the far end of the latency distribution

The flatness of the CDF curve is the difference between the 10 percentile and 90 percentile values. It informs how much the observed latencies are distributed where most of the latencies are observed. A smaller value of the flatness of the CDF curve indicates the latencies are more concentrated, and the application is performing more deterministically.

Overall, it is preferable for both the tail and flatness of the CDF curve to be as small as possible.

Conclusion

Holoscan flow benchmarking is currently available for four HoloHub C++ applications:

- endoscopy_tool_tracking
- multiai_endoscopy
- multiai_ultrasound
- ultrasound_segmentation

We plan to add Holoscan flow benchmarking for other C++ and Python sample applications for HoloHub in the future.

Holoscan flow benchmarking allows developers to evaluate the performance of Holoscan applications at scale. The article demonstrates a few common use-cases. Please let us know how you are leveraging Holoscan flow benchmarking and if you are experiencing any gaps or blockers for your applications. Please feel free to file issues on Github or contribute directly to HoloHub.

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Arm

Arm, AMBA, and Arm Powered are registered trademarks of Arm Limited. Cortex, MPCore, and Mali are trademarks of Arm Limited. All other brands or product names are the property of their respective holders. "Arm" is used to represent Arm Holdings plc; its operating company Arm Limited; and the regional subsidiaries Arm Inc.; Arm KK; Arm Korea Limited.; Arm Taiwan Limited; Arm France SAS; Arm Consulting (Shanghai) Co. Ltd.; Arm Germany GmbH; Arm Embedded Technologies Pvt. Ltd.; Arm Norway, AS, and Arm Sweden AB.

OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com