Newer Tools: ROCprof Trace Decoder and Omnistat

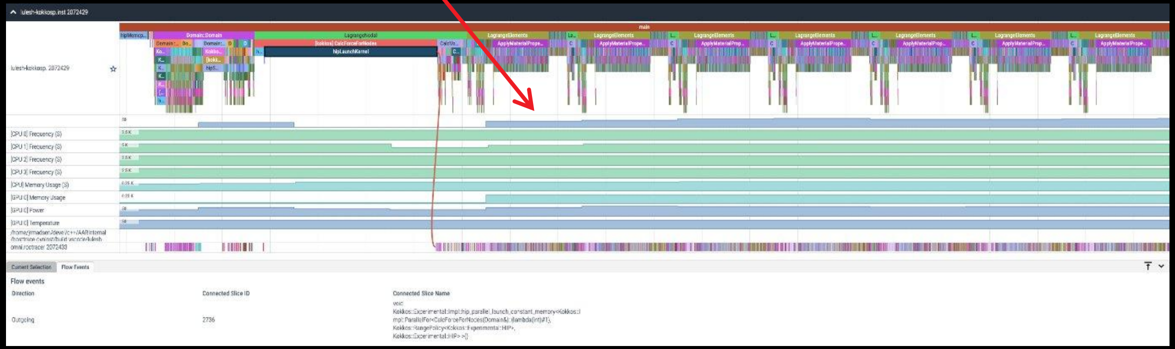
Presenter: Bob Robey
Oct 21-23, 2025
AMD @ Tsukuba University

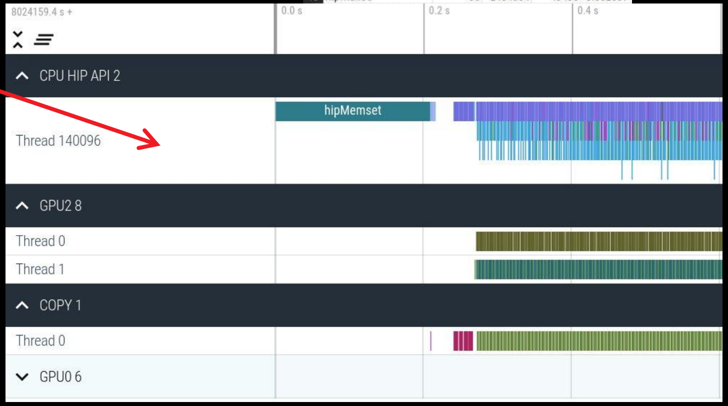
AMD
together we advance_

Role of ROCprof Compute Viewer and Trace Decoder?

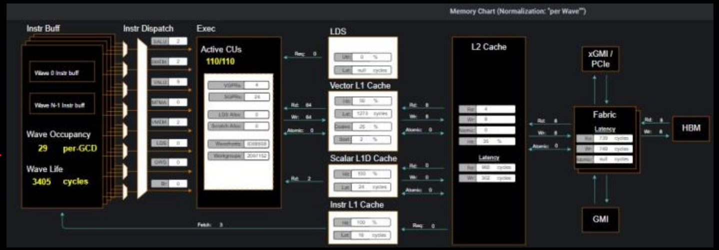
	A	B	C	D	E
1	Name	Calls	TotalDura	AverageN:	Percentage
2	hipMemcpyAsync	99	3.22E+10	3.25E+08	44.14872
3	hipEventSynchronize	330	2.42E+10	73394557	33.225
4	hipMemsetAsync	87	7.76E+09	89232696	10.64953
5	hipHostMalloc	9	5.41E+09	6.01E+08	7.415198
6	hipDeviceSynchronize	28	1.32E+09	47006288	1.805515
7	hipHostFree	17	1.05E+09	61534688	1.435014
8	hipMemcpy	41	8.11E+08	19791876	1.113161
9	hipL[illegible]Kernel	1856	58082083	31294	0.079676
10	hipStreamCreate	2	46380834	23190417	0.063625
11	hipMemset	2	18847246	9423623	0.025854
12	hipStreamDestroy	2	15183338	7591669	0.020828
13	hipFree	38	8269713	217624	0.011344
14	hipEventRecord	330	2520035	7636	0.003457
15	hipMalloc	30	1484804	49493	0.002037

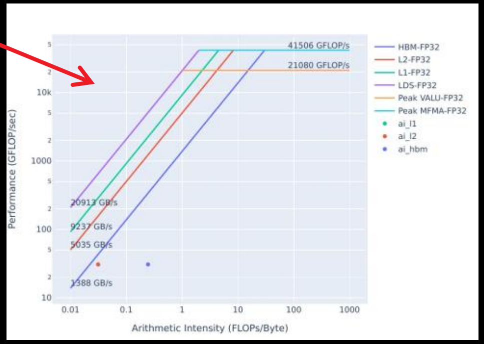
- Rocprofv3 enables basic profiling and hardware counters
- Rocprof-sys gives a system level view including CPU, MPI, OpenMP® and many others.





- Rocprof-compute provides a detailed overview of the performance of individual GPU kernels

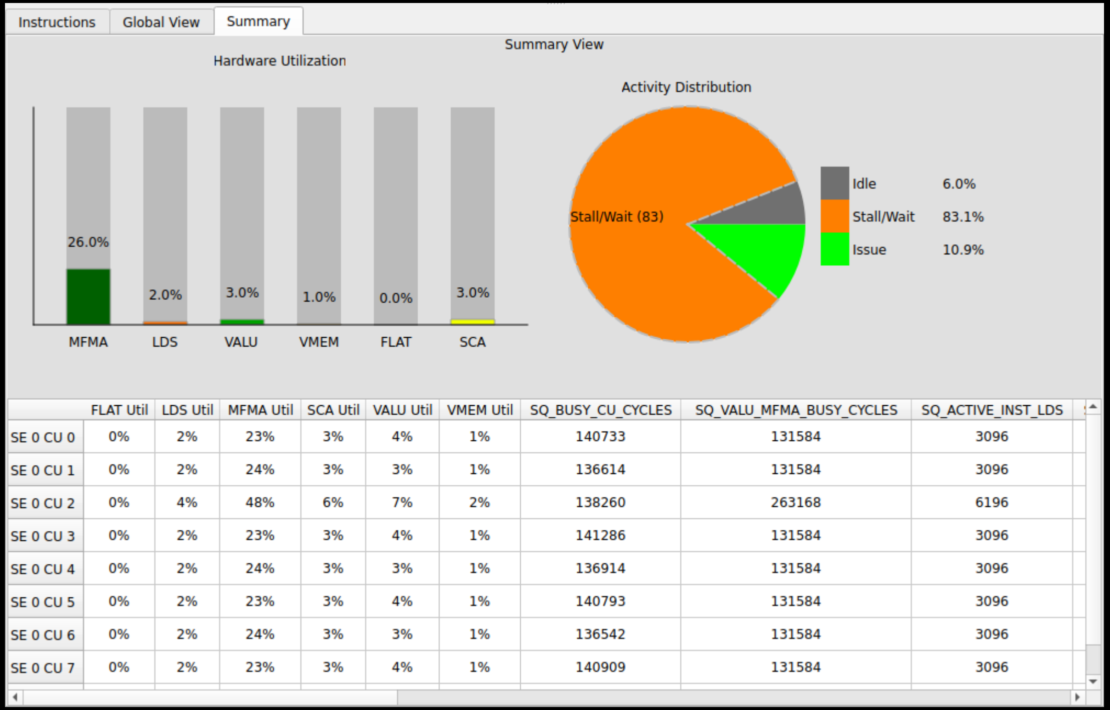




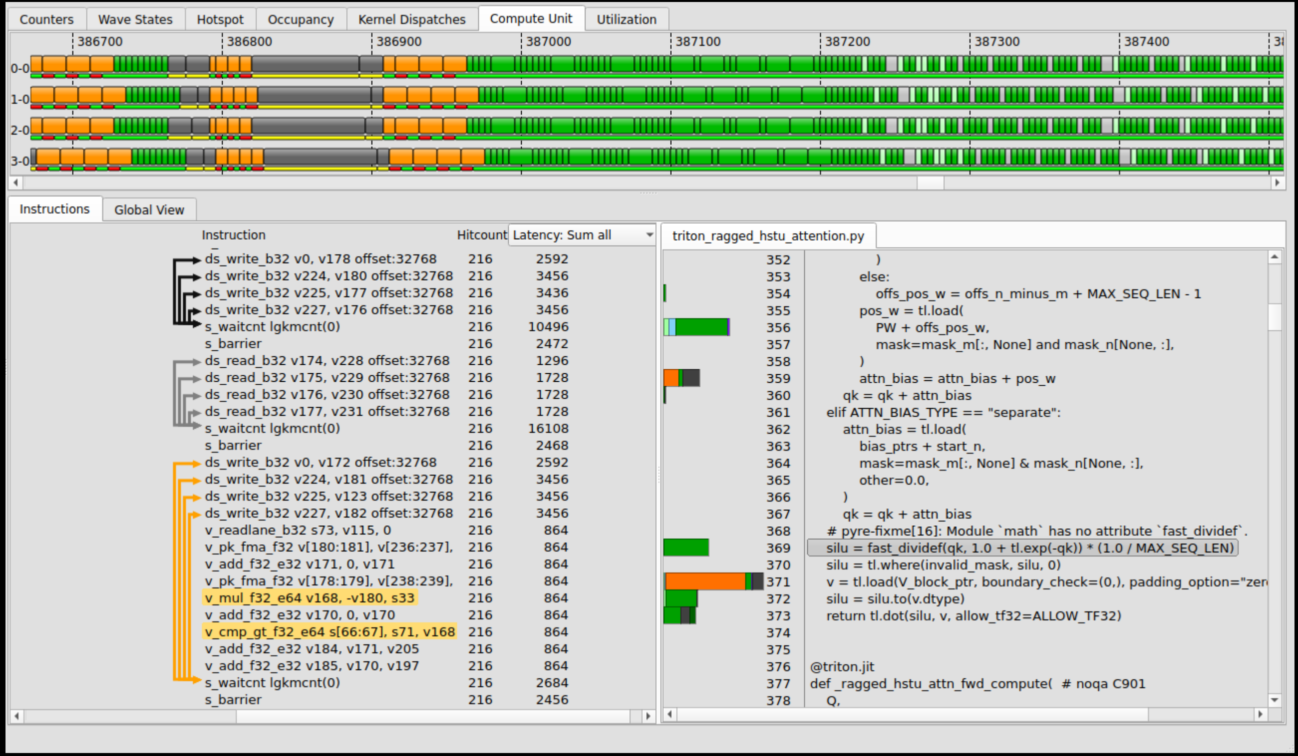
- But what has been missing is a tool that looks at performance ***within*** a kernel
  - What lines of code in the kernel are taking the most time?
  - Where are stalls happening?
  - What parts of the kernel are consuming resources such as LDS memory?

AMD together we advance_

ROCprof Compute Viewer – high-level view of kernels



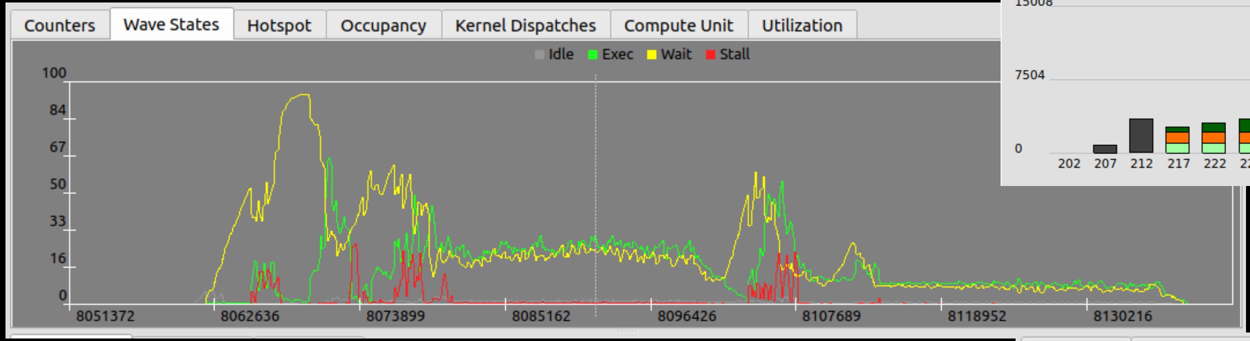
Summary View



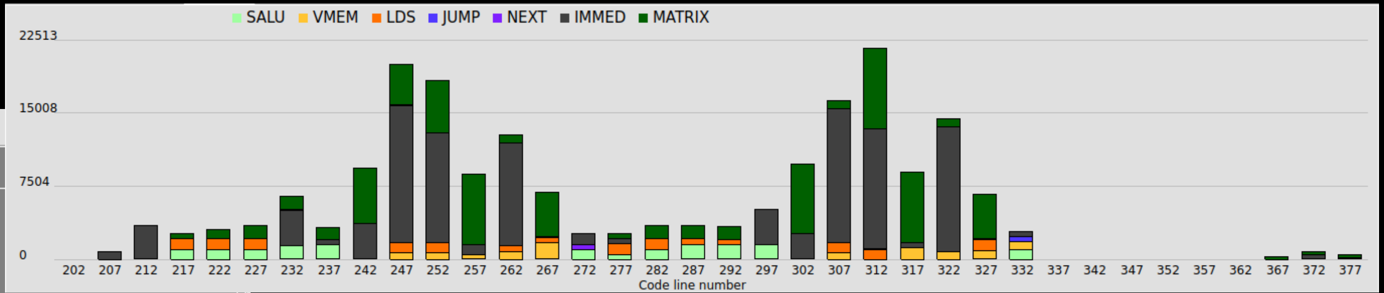
From https://rocm.docs.amd.com/projects/rocprof-compute-viewer/

Instructions View

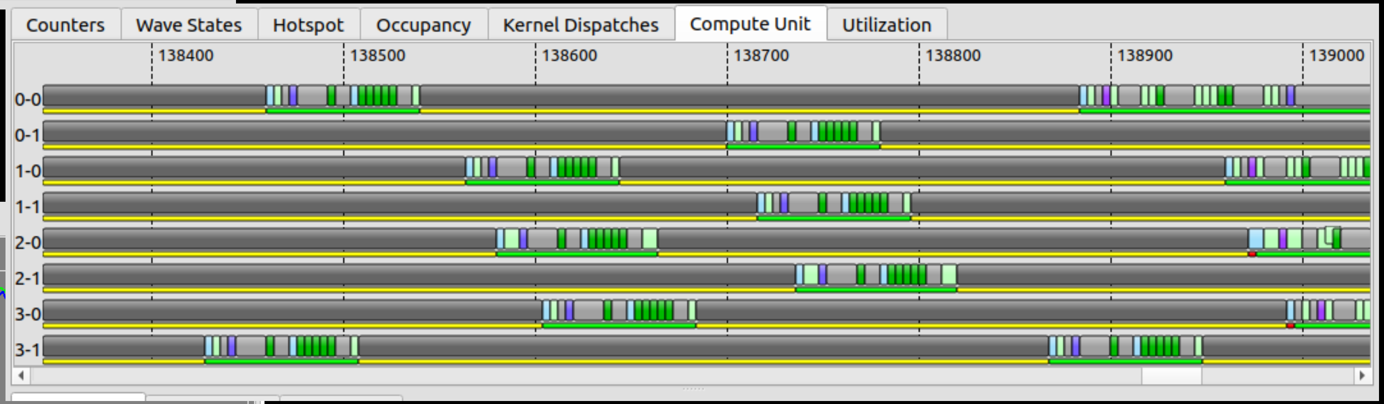
ROCprof Compute Viewer – Timeline views



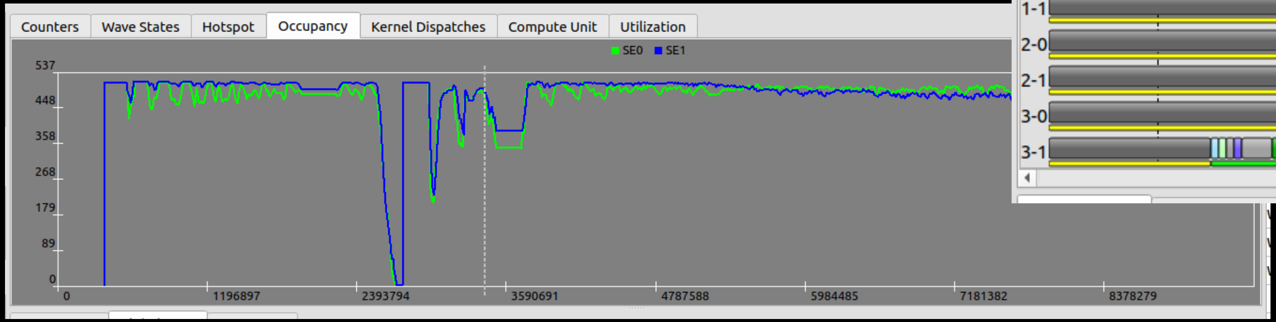
Wave States Timeline



Hotspot Timeline



Compute Unit Timeline



Occupancy Timeline

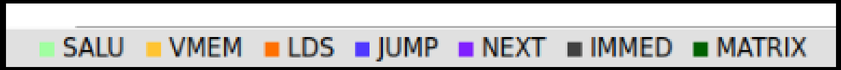
From https://rocm.docs.amd.com/projects/rocprof-compute-viewer/

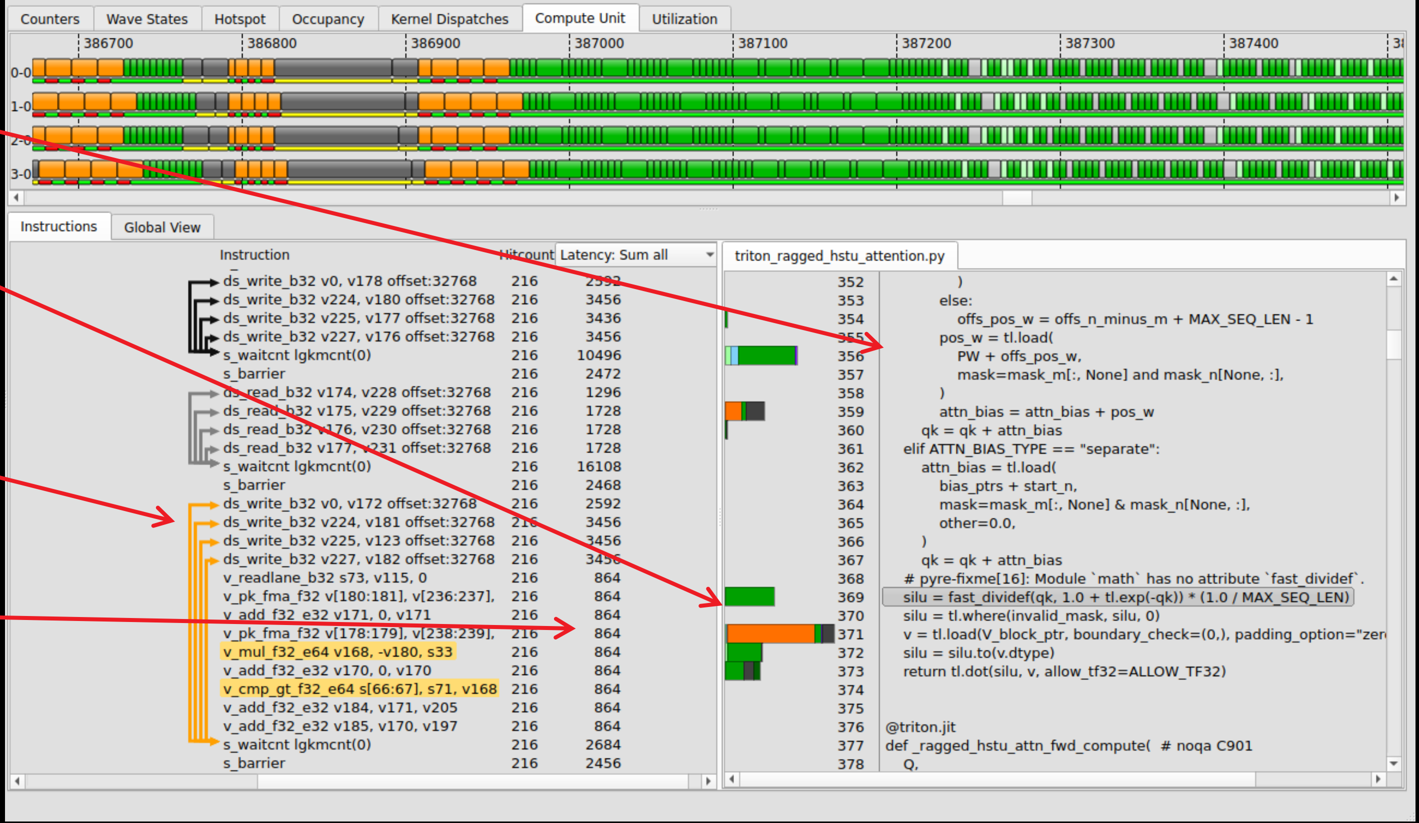
together we advance_

Instructions view – detail exploration

- We begin with the instructions view in the Rocprof Compute Viewer.
- The lines of code from the kernel are shown on the right.
- Just to the left of the kernel line is a bar indicating the resources used for that line.
- The assembler ISA for the source lines is shown on the left. This can quickly show whether a wide load instruction has been used or what are the compute instructions generated by the compiler. Also shown is the hit count and latency for each line
- Clicking on a source code or ISA line will reorient the other window to the corresponding line and left align the top timeline view of the compute unit.

Color coding for blocks





From https://rocm.docs.amd.com/projects/rocprof-compute-viewer/

Quickstart Usage Walkthrough

Generating trace decoder data

- Set up the environment – load the rocprofiler-sdk or rocm module depending on your system
```
module load rocprofiler-sdk
```
- Build the HIP code that you want to profile
```
cd ../HIP/vectorAdd
make vectoradd
```
- Generate the trace decoder data
```
rocprofv3 --att --att-activity 16 -d tracedecoder_vectorAdd -- ./vectoradd
```
- CLI Options
  - --att required to turn on trace decoder
  - --att-activity needed for summary (and maybe source code?). Frequency number is from 1 to 32.
  - Compile debug to get the source code included
- Tar up the directory specified for the data in the previous command
```
tar -czvf tracedecoder_vectorAdd.tgz tracedecoder_vectorAdd
```
- Copy the files to your local system
```
scp <hpc_system>:tracedecoder_vectorAdd.tgz tracedecoder_vectorAdd.tgz
```
- Clean up afterwards
```
make clean
rm -rf tracedecoder_vectorAdd
```

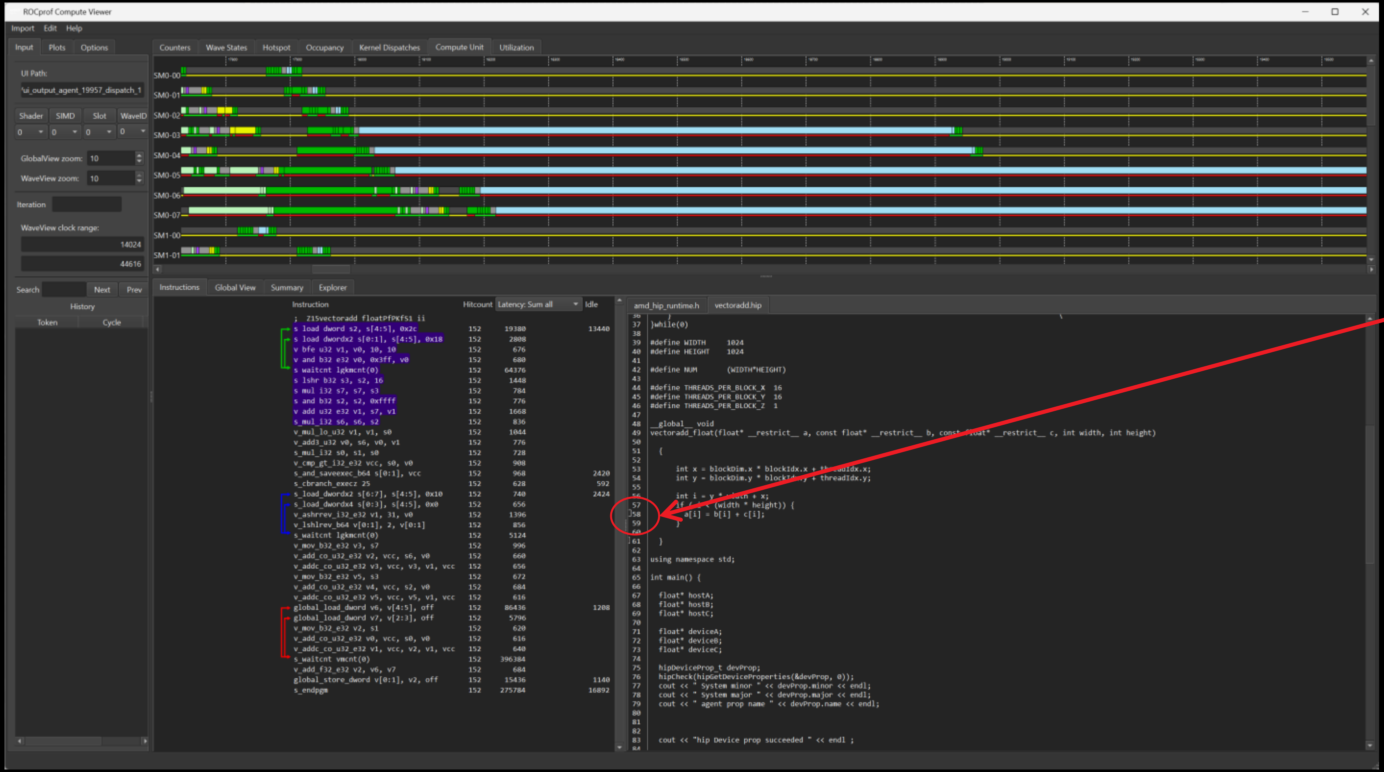

together we advance_

ROCprof Compute Viewer

- Untar the data that you have copied to your system

```
tar --xzvf tracedecoder_vectorAdd.tgz
```

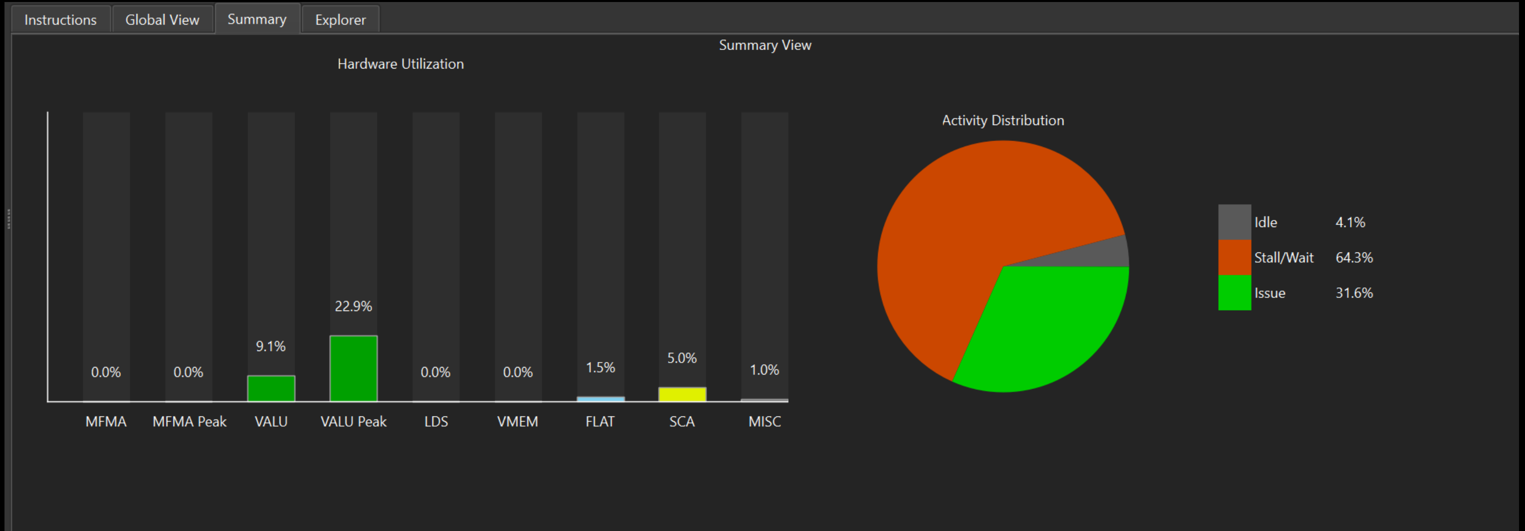
- Now start up ROCprof Compute viewer on your local system
- Use Import at the very top left and select one of the ui_output_agent* files in the tracedecoder_vectorAdd directory. Center the source code window, select vectorAdd.hip and scroll down to find the kernel.



- Select line 58 a[i] = b[i] + c[i]; and see what lines get highlighted in the left window.
- There is a very small rectangle on the left of line showing resource usage – we are not using much in this kernel.
- We compiled with -g -O2, so the lines may not be exact.

Summary View

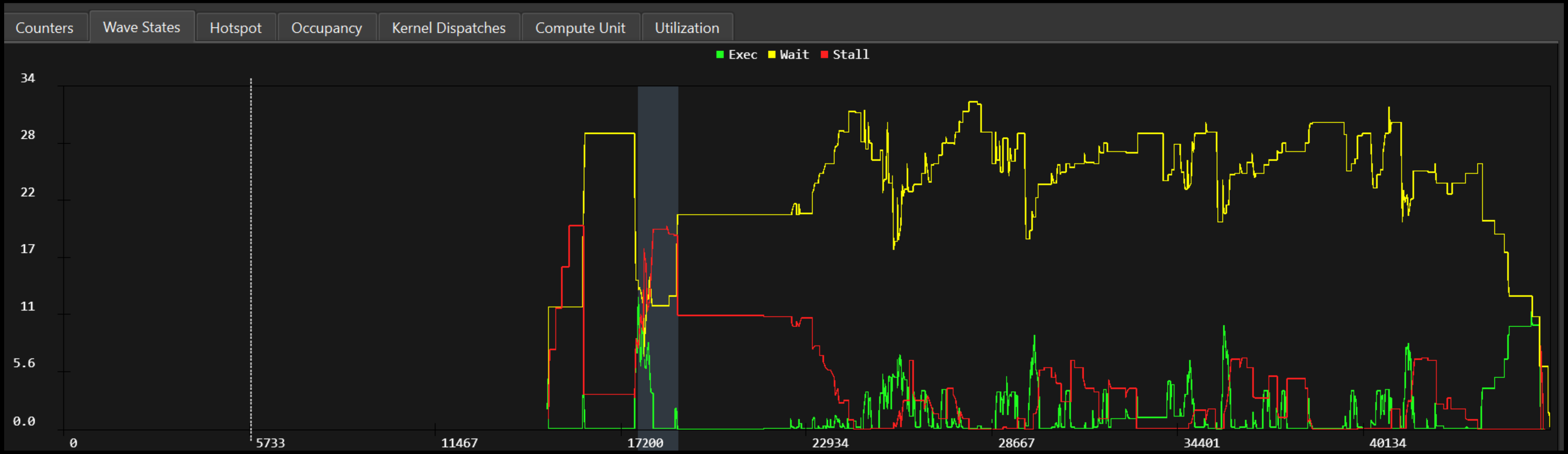
- Click on the Summary tab.
- We can see that we are mostly using VALU (Vector Arithmetic Logic Unit) instructions
- We are also mostly seeing Stall/Wait states
- We can conclude that we are memory bandwidth bound and that we could be doing a lot more compute work in this kernel.
- Based on this, let's investigate the wave state timeline to better understand what is happening.



	FLAT Util	LDS Util	MFMA Util	MISC Util	SCA Util	VALU Util	VMEM Util	SQ_BUSY_CU_CYCLES	SQ_VALU_MFMA_BUSY_CYCLES	SQ_ACTIVE_INST_VALU	SQ_ACTIVE_INST_LDS	SQ_ACTIVE_INST_VMEM	SQ_ACTIVE_INST_
Peak	3%	0%	0%	3%	15%	23%	0%	23296	0	5337	0	0	742
SE 0 CU 0	2%	0%	0%	1%	5%	10%	0%	30583	0	2952	0	0	492
SE 0 CU 1	1%	0%	0%	1%	5%	9%	0%	31770	0	2736	0	0	456
SE 0 CU 2	2%	0%	0%	1%	5%	9%	0%	31700	0	2880	0	0	480
SE 0 CU 3	1%	0%	0%	1%	5%	9%	0%	31772	0	2808	0	0	468
SE 0 CU 4	2%	0%	0%	1%	5%	9%	0%	30475	0	2808	0	0	468
SE 0 CU 5	1%	0%	0%	1%	5%	8%	0%	32220	0	2736	0	0	456
SE 0 CU 6	2%	0%	0%	1%	5%	9%	0%	31499	0	2880	0	0	480

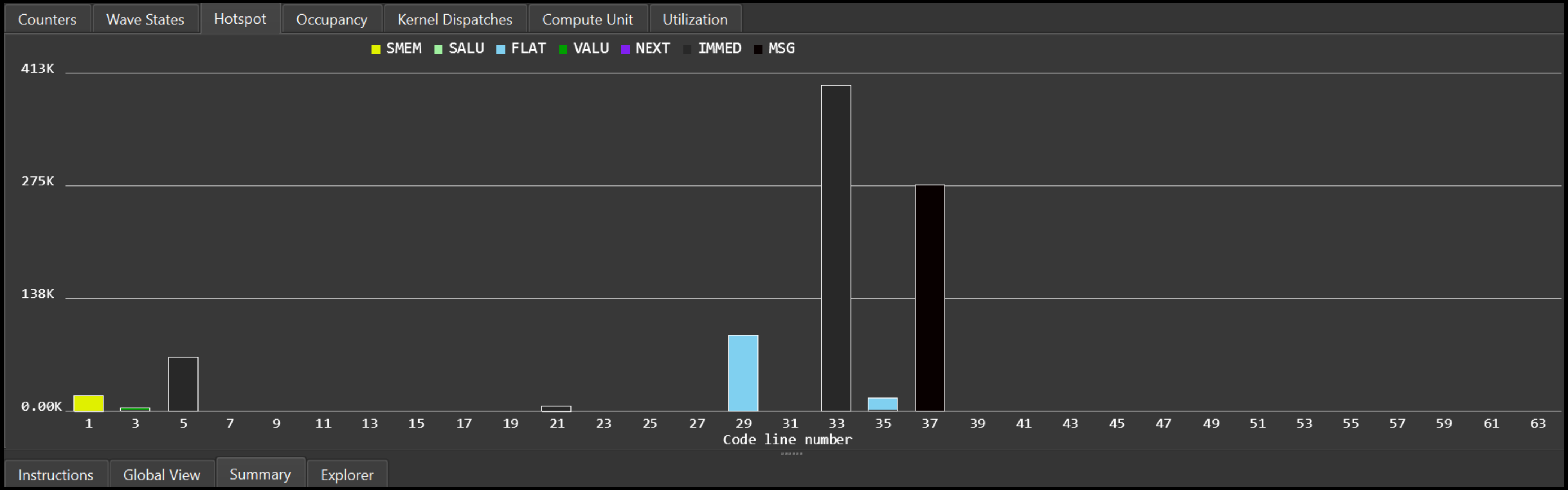
Wave States Timeline

- Click on the Wave States tab.
- We see that it is mostly waits rather than stalls. This tells us that we are likely just waiting on data loads.
- Compute (Exec) is low indicating that we could do more work.
- Let’s jump to the Hotspot view to see if we can learn something there.



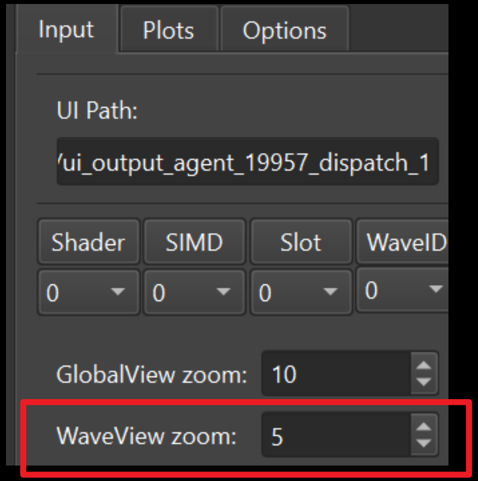
Hotspot Timeline

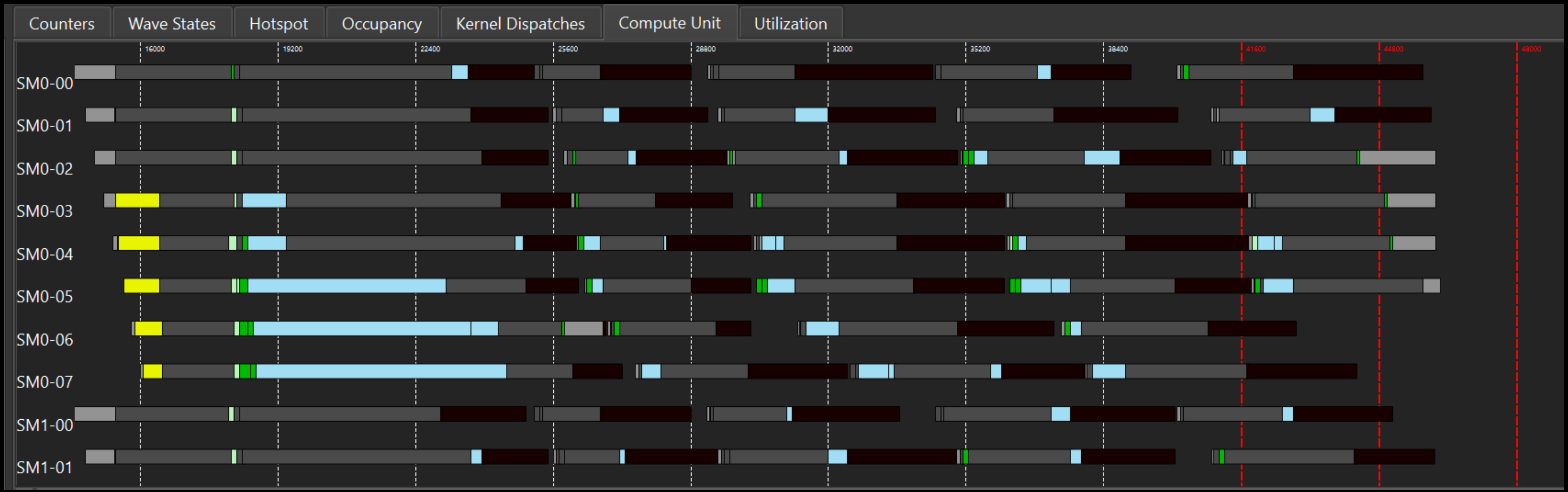
- We see an initial memory request and then fairly sparse activity.
- With a more complex kernel, we should see where LDS usage, matrix and other instruction types occur.
- Let's switch to the Compute Unit view to get a big picture of what is going on at that level.



Compute Unit Timeline

- Click on Compute Unit. Then go over to the left control panel and change the WaveView zoom from 10 to 5 so that we can see the whole timeline.
- We clearly see the initial memory load requests in yellow and then scattered VALU instructions in light blue.

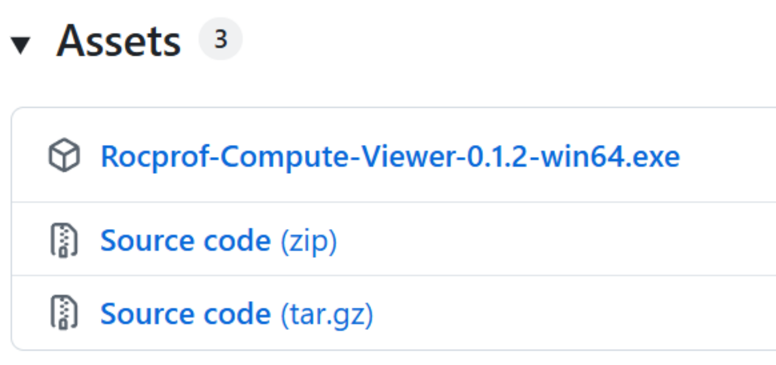


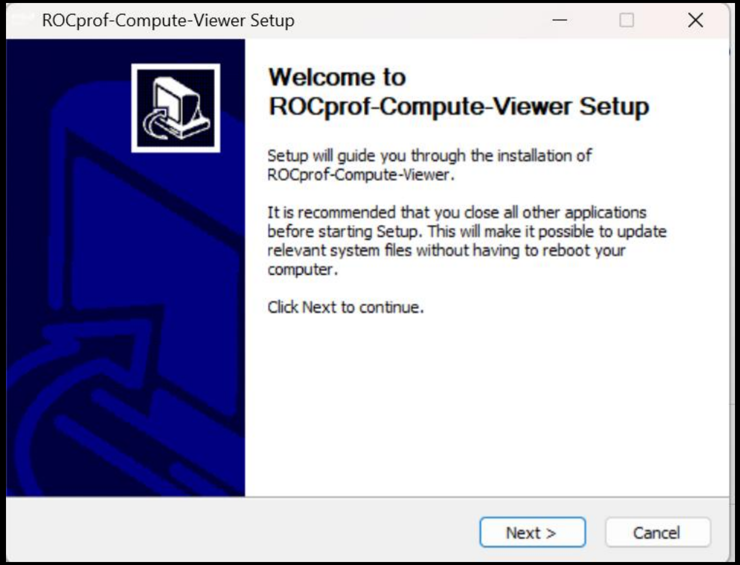


Installation Instructions

Installing ROCprof Compute Viewer

- Check for releases of HPC Compute Viewer
- https://github.com/ROCm/rocprof-compute-viewer/releases
- For Windows
- , there is a pre-built version for download
  - Uses a Windows setup tool
- To build from source, see
- https://rocm.docs.amd.com/projects/rocprof-compute-viewer/en/amd-mainline/install/installation.html#install-viewer
- Linux®, Mac OS, Windows Subsystem for Linux (WSL) and Windows® native
- Uses Qt 5 or 6





Setting up Trace Decoder

Nothing needed on ROCm 7.0+ !!

Trace Decoder is a part of ROCprofv3

- We have installed on ROCm version 6.4.1 with some effort.
  - Install recent rocprofiler-sdk from source to a different location (/opt/rocmplus-6.4.1/rocprofiler-sdk)
  - Install aqlprofile to same location
  - Install tracedecoder binary to same location
  - Set paths to the new rocprofiler-sdk directory before the standard ROCm location
  - Need to override default gfx codes since some do not exist before ROCm 7.0
  - Install to prior ROCm versions may break in the future
  - Cannot install prior to ROCm 6.2
- May consider using a ROCm 7.0 container to get early access as well

Setup pre-7.0 ROCm version step 1

- Requires system have libdw installed
- Setup script with these steps at

  ```
  git clone https://github.com/AMD/HPCTrainingDock
  ```

  - File: HPCTrainingDock/tools/scripts/rocprofiler-sdk_setup.sh
  - Check script for latest instructions
- Set INSTALL_PATH to /opt/rocmplus-6.4.1/rocprofiler-sdk or other directory

- Install trace decoder

```
wget https://github.com/ROCm/rocprof-trace-decoder/releases/download/0.1.2/rocprof-trace-decoder-manylinux-2.28-0.1.2-Linux.tar.gz
tar -xzvf rocprof-trace-decoder-manylinux-2.28-0.1.2-Linux.tar.gz
mv rocprof-trace-decoder-manylinux-2.28-0.1.2-Linux/opt/rocm/lib/librocprof-trace-decoder.so $INSTALL_PATH/lib
rm -rf rocprof-trace-decoder-manylinux-2.28-0.1.2-Linux
```

Setup pre-7.0 ROCm version step 2

- Install rocprofiler-sdk
- AMDGPU_GFXMODEL is set to “gfx90a;gfx942” to override default gfx model list. Use what you need here.

```
git clone –b develop https://github.com/ROCm/rocm-systems.git rocm-systems-source

cmake -B rocprofiler-sdk-build                \
      -DCMAKE_INSTALL_PREFIX=${INSTALL_PATH}  \
      -DGPU_TARGETS=\"${AMDGPU_GFXMODEL}\" \
      -DOPENMP_GPU_TARGETS=\"${AMDGPU_GFXMODEL}\" \
      -DCMAKE_PREFIX_PATH=/opt/rocm-${ROCM_VERSION} \
       rocm-systems-source/projects/rocprofiler-sdk

cmake --build rocprofiler-sdk-build --target all --parallel 8
cmake --build rocprofiler-sdk-build --target install

rm -rf rocprofiler-sdk-source rocprofiler-sdk-build
```

Setup pre-7.0 ROCm version step 3

- Install aqlprofile

```
cmake  -B aqlprofile-build                         \
       -DGPU_TARGETS="${AMDGPU_GFXMODEL}"    \
       -DCMAKE_PREFIX_PATH=/opt/rocm-${ROCM_VERSION}/lib:/opt/rocm-${ROCM_VERSION}/include/hsa \
       -DCMAKE_INSTALL_PREFIX=$INSTALL_PATH  \
       rocm-systems-source/projects/aqlprofile

cmake --build aqlprofile-build --target all --parallel 8
cmake --build aqlprofile-build --target install

rm -rf rocm-systems-source aqlprofile-build
```

Omnistat

Omnistat

- Omnistat is primarily for HPC cluster administrators
- Has both a system-wide and a user-mode
- Metrics include:
  - GPU utilization
  - High-bandwidth memory (HBM) usage
  - GPU power
  - GPU temperature
  - GPU clock frequency
  - GPU memory clock frequency
  - Inventory information:
    - ROCm driver version
    - GPU type
    - GPU vBIOS version
- Optional metrics
  - RAS information (error counts per GPU block)
  - GPU power caps
  - GPU throttling events
  - Host network traffic (received/transmitted)

Sample output – GPU Usage Summary

Omnistat: Where to get more information

- AMD Research project and is not part of the ROCm software stack.
- Online documentation
  - https://rocm.github.io/omnistat
- Source code
  - https://github.com/ROCm/omnistat

- Uses either Prometheus client and server or VictoriaMetrics server for scalable collection of metrics
- Uses Grafana for visualizing data

Disclaimer

AMD