Tracing and debugging heterogeneous CPU-GPU systems

Arnaud Fiorini

August 20, 2019

Polytechnique Montreal

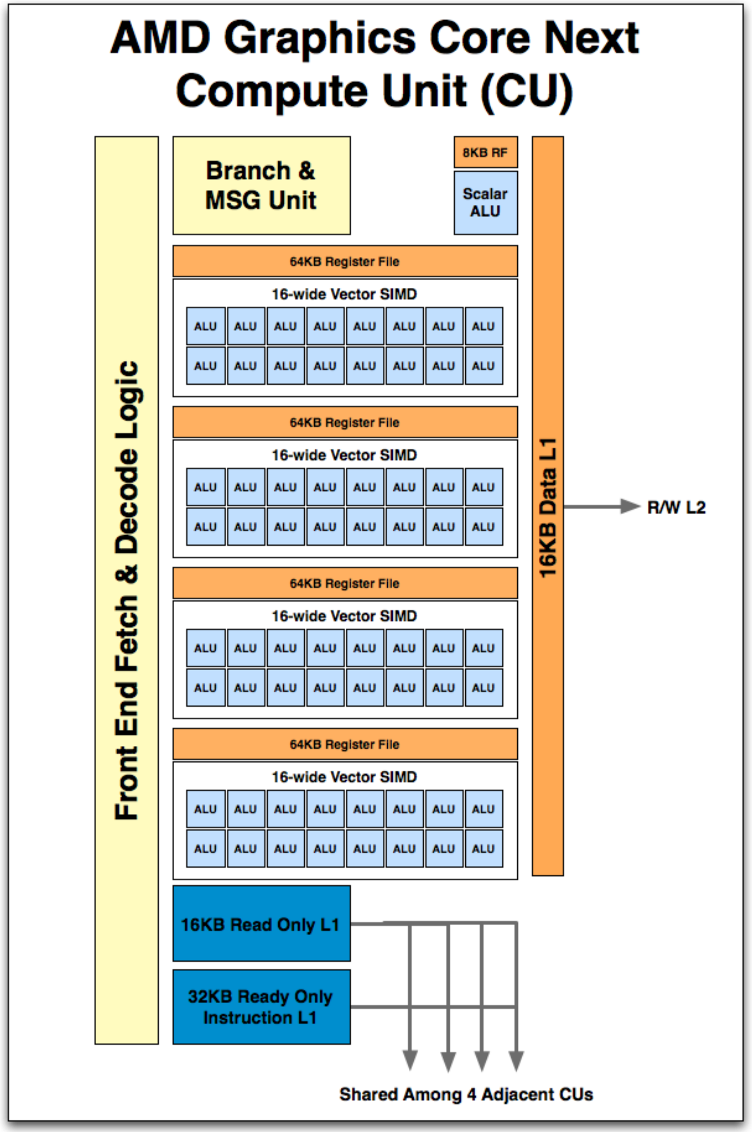
DORSAL Laboratory

Agenda

I. Tracing and profiling of CPU-GPU systems
   1. Introduction to GPU architecture and ROCm
   2. Tracing GPUs
   3. Demo

II. Debugging CPU-GPU systems
   1. Review of CPU debugging
   2. Challenges of GPU debugging

Tracing and profiling of CPU-GPU systems - Introduction



Tracing and profiling of CPU-GPU systems - Introduction

- To execute a program on a GPU, you need a kernel.
- A kernel is a function that is executed by the GPU many times concurrently.
- How this function is written depends mostly on the interface used : OpenCL, CUDA, C++ Parallel STL …

Tracing and profiling of CPU-GPU systems - Introduction

- An example :

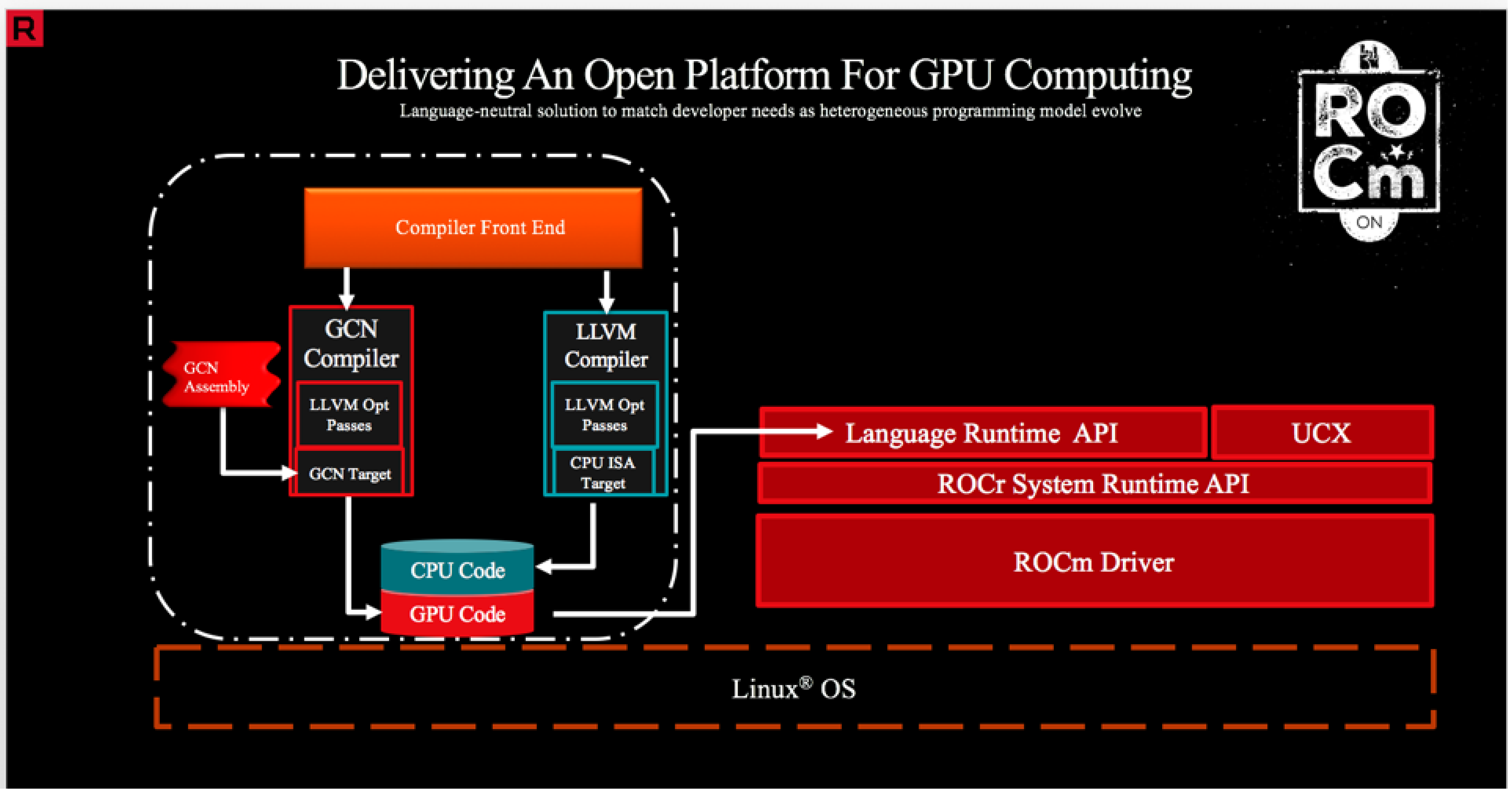
```
__kernel void saxpy(__global float *src, __global float *dst, float factor)
{
  long i = get_global_id(0);
  dst[i] += src[i] * factor;
}
```

Tracing and profiling of CPU-GPU systems - Introduction

If something goes wrong inside this function, or the execution is too slow…

We want to understand why.

Tracing and profiling of CPU-GPU systems - Introduction



Tracing and profiling of CPU-GPU systems - Introduction

- ROCr : Radeon Open Compute runtime

- It works for GPUs but it can also work for any other specialised hardware using the Heterogeneous System Architecture (HSA).

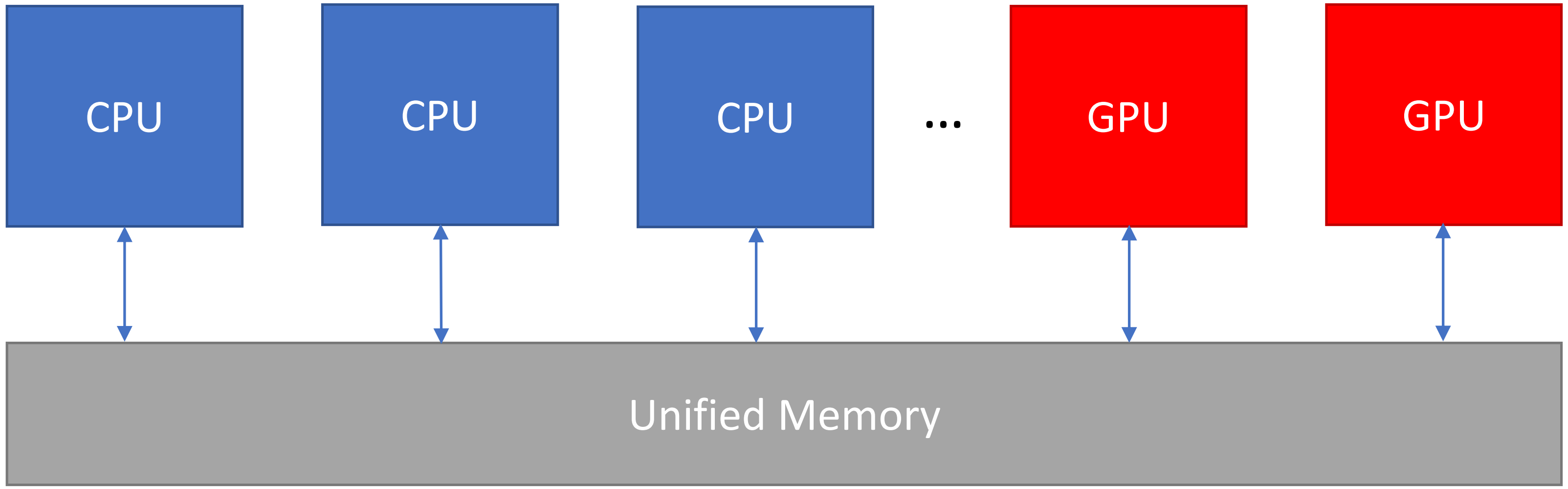
Tracing and profiling of CPU-GPU systems – Tracing GPUs

Why ROCm ?

- It is Open Source.
- It uses user queues by sharing memory between the CPU and the GPU.

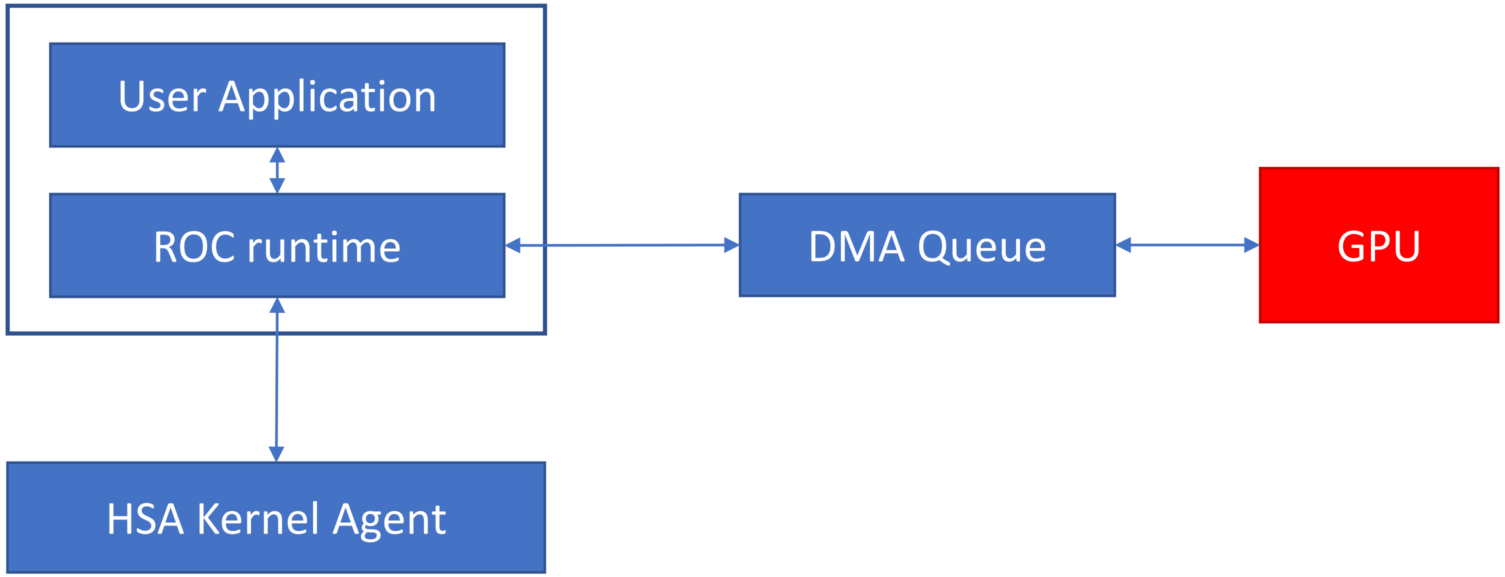
Tracing and profiling of CPU-GPU systems – Tracing GPUs

One of the key features of HSA is the heterogeneous Unified Memory Access :



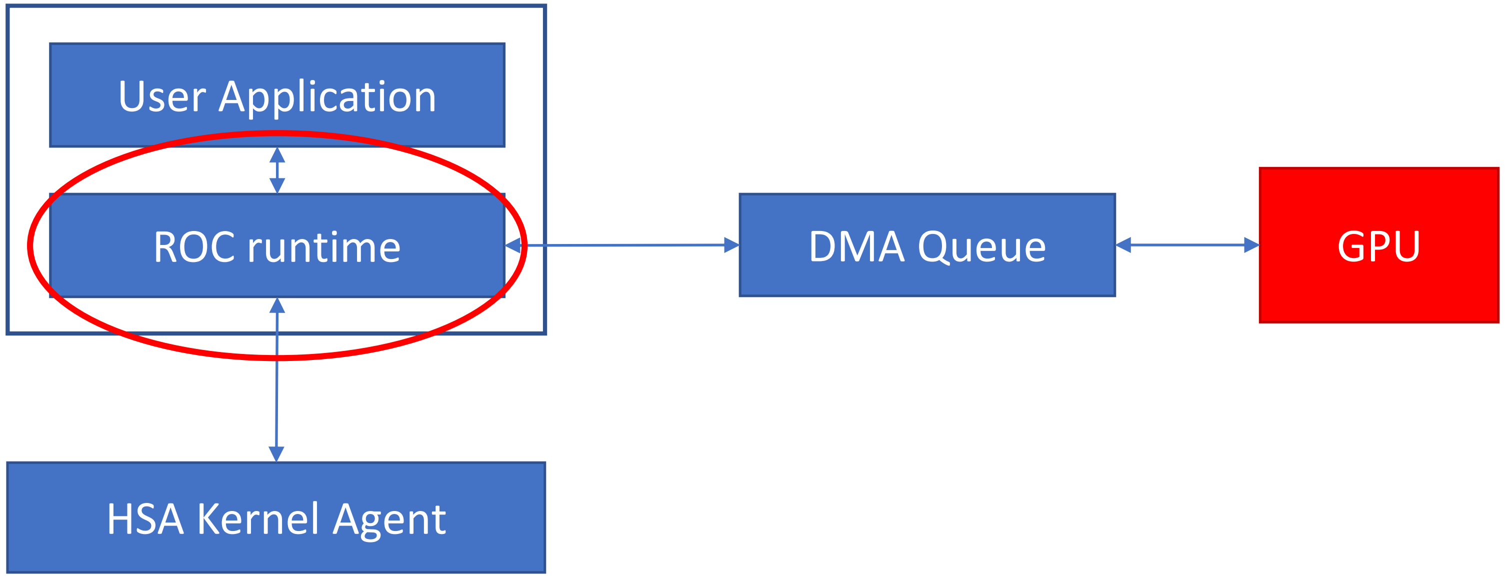
Tracing and profiling of CPU-GPU systems – Tracing GPUs

This feature is essential for user queues :



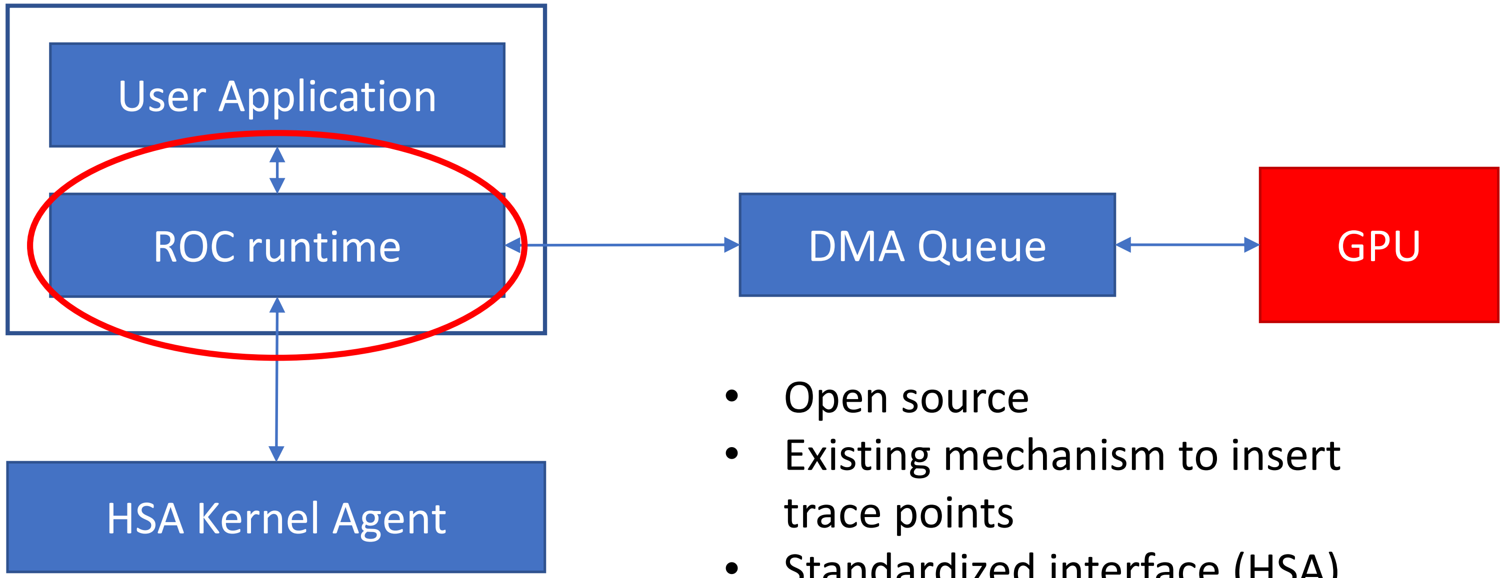
Tracing and profiling of CPU-GPU systems – Tracing GPUs

This feature is essential for user queues :



Tracing and profiling of CPU-GPU systems – Tracing GPUs

This feature is essential for user queues :



- Open source
- Existing mechanism to insert trace points
- Standardized interface (HSA)

Tracing and profiling of CPU-GPU systems – Tracing GPUs

How does it work ?

- It is possible to insert code before and after each calls to the ROC runtime by using the HSA_TOOLS_LIB environment variable.
- This environment variable is used by the ROC runtime to load different tools.
- The ROC runtime opens the library using dlopen.

Tracing and profiling of CPU-GPU systems – Tracing GPUs

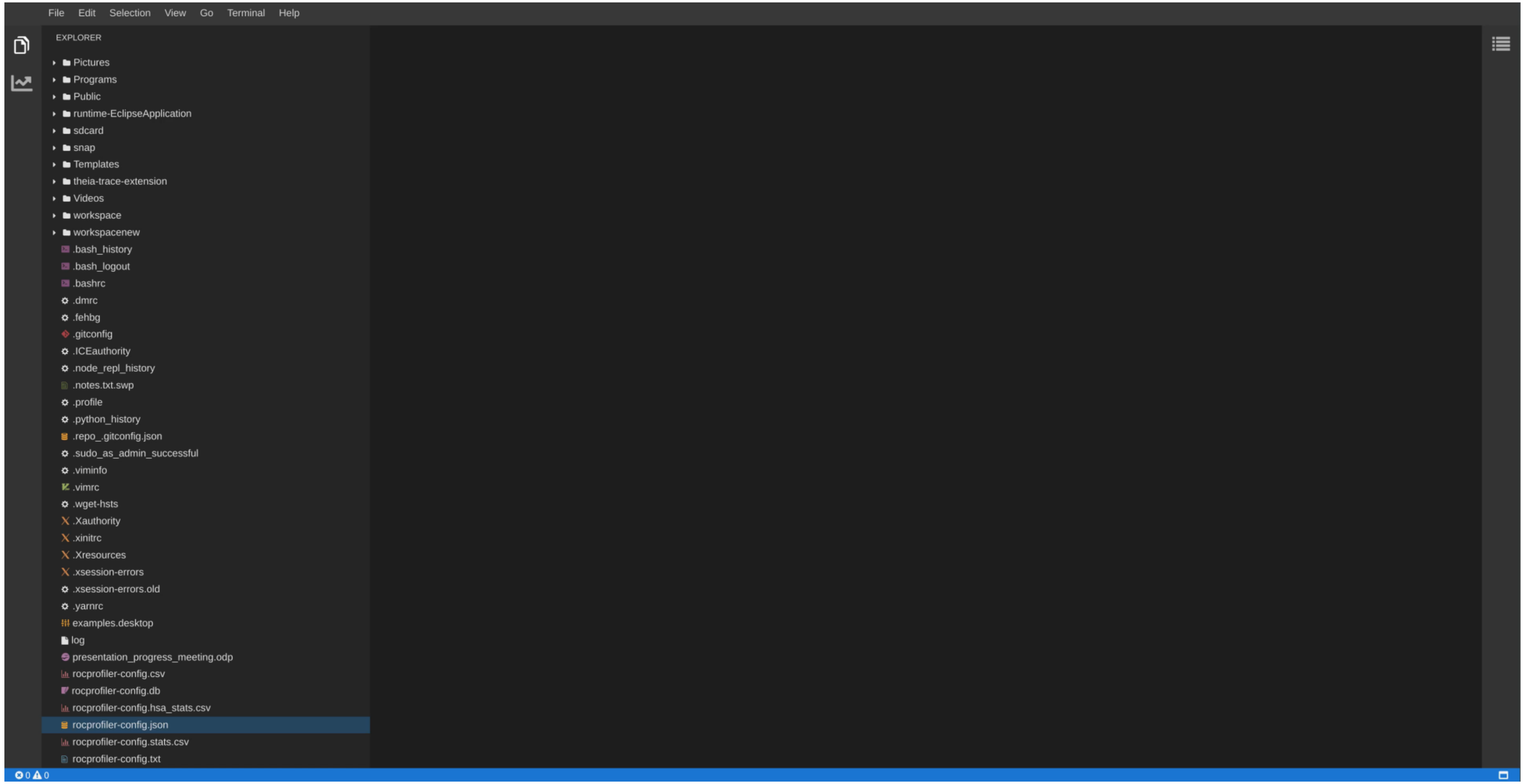
- This work has already been done by AMD and is open source :
  https://github.com/ROCm-Developer-Tools/rocprofiler
  https://github.com/ROCm-Developer-Tools/roctracer

- AMD has released a few other libraries and tools thanks to their Radeon Open Compute initiative.

Tracing and profiling of CPU-GPU systems – Tracing GPUs

- The tracepoints do not use any tracing tool to write the events.
- The resulting trace is then stored with the TraceEvent format (JSON).
- This trace is then analyzed by Trace Compass and visualized using Theia.

Tracing and profiling of CPU-GPU systems – Demo



Debugging CPU-GPU systems – Review of CPU debugging

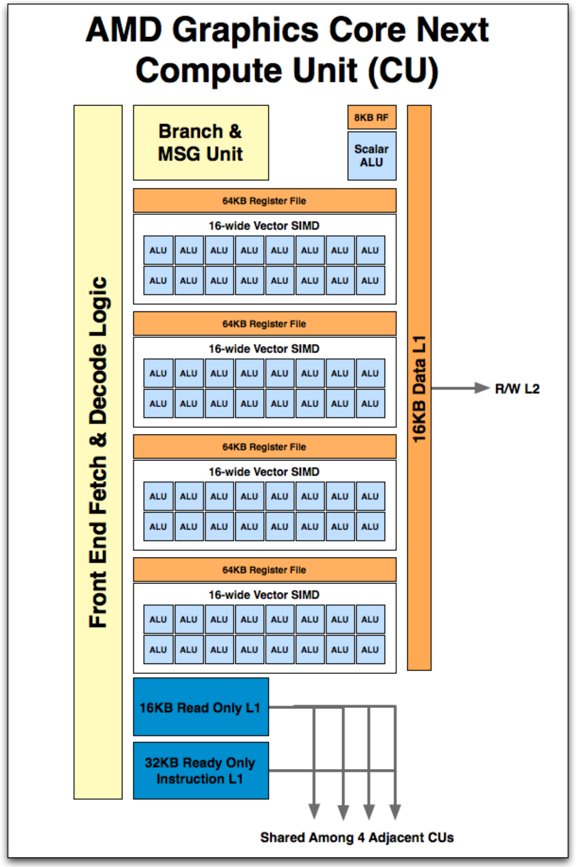
The most essential feature of a debugger is placing a breakpoint.

A review of how it works (in general):

1. Read the instruction at the address and save it
2. Write a trap instruction
3. The program runs into the trap instruction
4. Then the debugger checks the value of the instruction pointer to check if it’s a breakpoint
5. To continue, it writes the instruction that was saved
6. The debugger single-steps the instruction and changes the instruction again

Debugging CPU-GPU systems – Challenges of GPU debugging

In a GPU, each program executes multiple times in different SIMD processors :



Debugging CPU-GPU systems – Challenges of GPU debugging

Because of the complexity of GPU hardware, it becomes difficult to specify what is a thread in the context of debugging.

Architecture concepts

- Compute Unit
- SIMD Vector Unit
- ALU

Logical concepts

- Wavefront
- Thread
- Work-item

Debugging CPU-GPU systems – Challenges of GPU debugging

Because of the complexity of GPU hardware, it becomes difficult to specify what is a thread in the context of debugging.

Architecture concepts

- Compute Unit
- SIMD Vector Unit
- ALU

Logical concepts

- Wavefront
- Thread
- Work-item

64 threads / wavefront

Debugging CPU-GPU systems – Challenges of GPU debugging

Thread preemption

- With instruction-level preemption, it is possible to debug on one GPU.
- Otherwise, debugging requires an additional separate GPU for display while debugging.

Thank you for listening !

Questions ?

References

- https://github.com/RadeonOpenCompute/ROCm
- https://rocm-documentation.readthedocs.io/en/latest/
- http://www.hsafoundation.com/
- HSA Runtime Programmer’s Reference Manual, Version 1.2
- HSA Programmer's Reference Manual, Version 1.2
- HSA Platform System Architecture Specification, Version 1.2
- https://github.com/ucb-bar/opencl-kernels/blob/master/saxpy/kernel.cl
- https://medium.com/@smallfishbigsea/basic-concepts-in-gpu-computing-3388710e9239
- https://www.techpowerup.com/gpu-specs/docs/amd-gcn1-architecture.pdf