CPU-GPU Communication for tools

https://bit.ly/swt-cpu-gpu

Problem space

- Data has to get back to the CPU at some point
- Several models
  - Wait for the end of the kernel
  - Triggering from the CPU side periodically
  - Trace buffers are complete, have to empty them
- Data format
- What’s the synchronization?
- AMD has interest(?) for a library(?) to interact with the GPU
  - Idea phase
- Could this also be used at an application level ?

Technical Issues

- Have to inject HSA packets
- CPU load
- SDMA unit can probably not be used during a kernel execution
- Coarse-grained : *might* work for different pages
- S_SENDMSG semantics & driver-side implementation
  - who is on the receiving end of the message
  - is message and handler customizable (an example would be great)
- Trap handler implementation ? https://martty.github.io/posts/radbg_part_4/

How could we do it?

- Coarse-grained data
  - Guaranteed to be synchronized at kernel bounds
- Fine-grained data
  - Changes are reproduced on the host
  - Memory model? -> ROCm doc
- PCIe atomics (int64_t)
  - HSA describes this (“fine-grained” memory) doorbell
- Not all memory transfers use the SDMA engine (< 32GB/s)
  - Faster cards (MI200, MI300, …) use “blit kernels” to transfer data hidden from the user
  - Documented? Probably not as they’re not open source
  - Determined by the runtime
  - They might show up through the HSA interception API

- Ben’s idea : fixed size buffer, insert packets in the HSA queue
  - Pre-allocate buffer (say, per kernel)
  - When it is known to be full, signal the GPU through a special HSA packet
  - Caveat : granularity is for multiple kernels
- What about a single kernel with high throughput
  - Can we trigger this *during* execution?
  - What would be the implications of stopping a kernel (counters, ..)
  - Multi-buffering (isomorphic to ring buffers, just different granularities)

Implementation directions

- How to handle memory copies : managed memory or actual shared memory
- Focus shared (MI300A-style) memory first, *might* work on older architectures
- Semantics for buffering
  - Used in rocprofiler apparently
  - Can this be a public interface
- API format
  - Raw/”packet”/buffer based
  - Lowest possible layer (so “raw”?), unstructured bytes
  - Potential descriptors (format string?)
  - API similar to a file system (r/w, mmap model)
  - Ben has been thinking about doing this for counters
- Instrumentation tools include heap memory for its perf data, kernel is modified to access this allocated data, runtime handles periodic/triggered copies to the host
- In line with AMD interests

Current rocprofiler-sdk impl

- Copy at the end of the kernel
- Kernel serialization (but can probably be avoided ?)