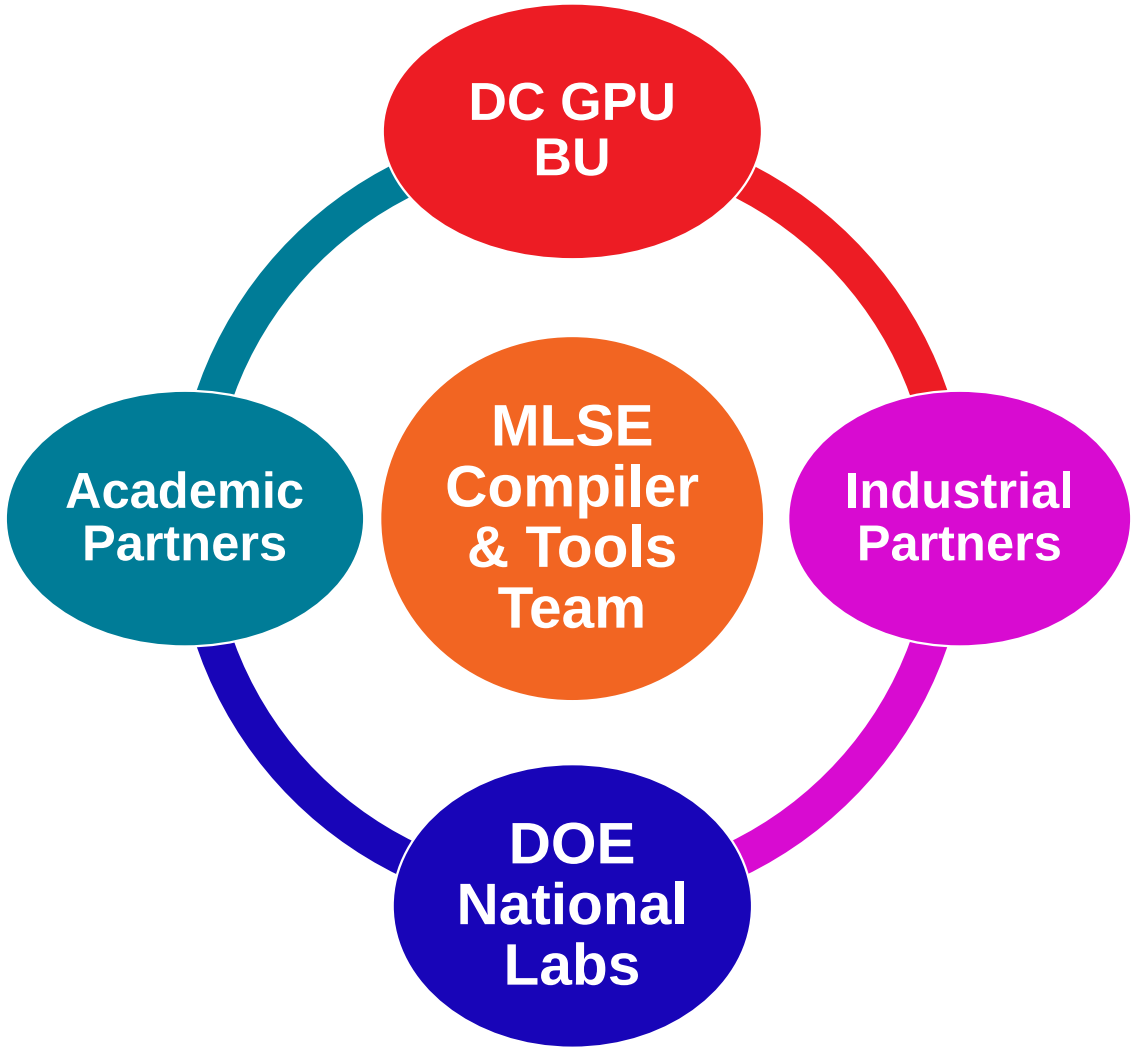
AMD ROCm Tools Progress: Profiler/Tracer, Debugger, OMPT, OmniTools and Binary Instrumentation Support

Timour Paltashev
with help of Ammar Elwazir, Tony Tye, Laurent Morichetti, Dhruva Chakrabarti, Zoran Zaric and Nursultan Kabylkas

Scalable Tools Workshop'2023

AMD
together we advance_

AMD ROCm Tools Ecosystem Essentials



- MLSE in cooperation with DC GPU BU provides funding of both 3rd party tools and libraries projects
- MLSE provides the equipment and technical cooperation with biweekly status review with tools ecosystem 3rd party contributors
- MLSE Tools projects are in three major domains
  - HPC/ML Code Debugging
  - HPC/ML Code Profiling and Tracing
  - HPC/ML Code Instrumentation (source&binary)
- Both academic and industrial partners involved (including indirect ones) in collaboration with National Labs
- Mostly open-source projects except integration with commercial product

AMD ROCm Tools Landscape

- **ROCgdb Debugger**
  - **GDB 13 based**
  - **LDS Variables**
  - **Navi 3.X support**
  - Linaro Forge DDT
  - Theia IDE
- **ROCdbgapi Debugger API**
  - ROCm Debug Agent
  - ROCgdb
  - Perforce TotalView
  - HPCToolkit
- **ROCprofilerV1 ROCtracerV1**
  - Exa-PAPI, LIKWID
  - ROCprof V1, TAU
  - HPCToolkit, OmniPerf
  - Score-P, OmniTrace
  - Caliper, Trace Compass
- **ROCprofiler V2**
  - Work in progress with ROCprof V2
  - Single unified library
  - Fixing issues with V1
  - Clients, services & buffers introduced
  - New functions and PC Sampling in V2 API
- **M-R GPU ISA XML Spec**
  - Joint effort with RTG Tools for Instinct and Radeon Products
  - For binary instrumentation and ISA simulators
  - Autogenerated from HW definition files
  - Machine-readable XML format with API
  - Executable with verified semantics
- **OmniTools AMD Research**
  - **OmniPerf** Support for MI100/ MI200/ MI300
  - **OmniPerf** Roofline And AMDGPU Perf. Models
  - **OmniTrace** Profiling & Tracing Tool
  - **OmniTrace:** Collection of runtime & system-level metrics
- **OMPT Support**
  - Exa-Tracer: ROCm + LTTng
  - HIP capture /instrmt /play
  - **HIP** Program textbook **2nd ed.**

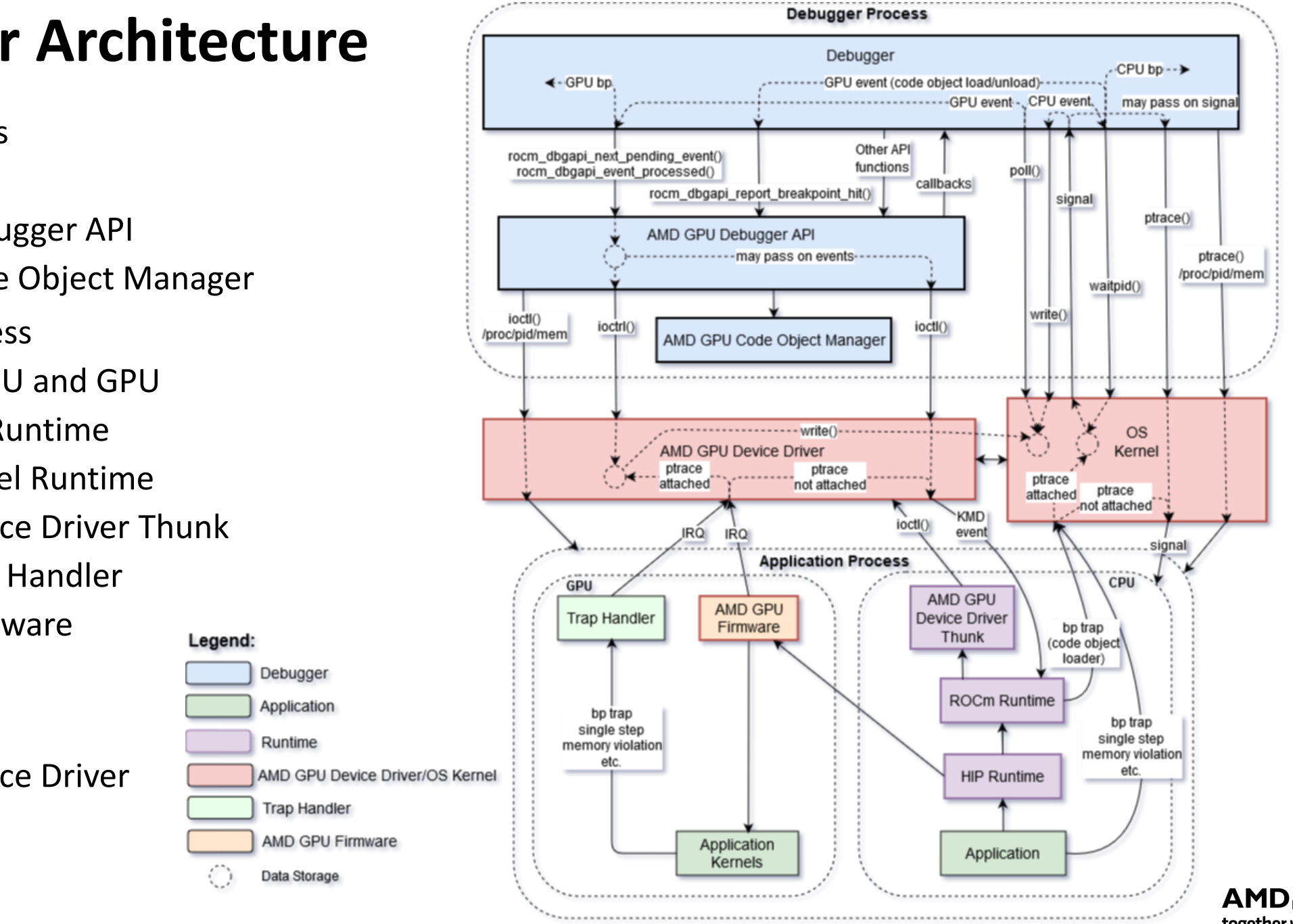
Accelerated Computing with HIP

- 226 pages
- Guides GPU programmers in developing HIP programs for the ROCm platform
  - Reasons through real-world problems, breaking them down into independent parts so that GPUs can be used to solve them efficiently
  - Tours GPU hardware design and demonstrates how to effectively leverage its unique hardware features to optimize software performance
  - Describes the ROCm ecosystem, and use of libraries and multiple GPUs
- Available from B&N and Amazon
  - Paperback $39.95, electronic format $11.99
- Associated Online course and e-learning coming soon

ROCm Tools: Debugger Status

Debugger Architecture

- Debugger Process
  - Gdb Debugger
  - AMD GPU Debugger API
  - AMD GPU Code Object Manager
- Application Process
  - Application: CPU and GPU
  - HIP Language Runtime
  - ROCm Low Level Runtime
  - AMD GPU Device Driver Thunk
  - AMD GPU Trap Handler
  - AMD GPU Firmware
- Linux Kernel
  - Linux Kernel
  - AMD GPU Device Driver



Debugger: Features Progress

- ROCm 5.6 (June 2023)
  - Rebased on GDB 13
  - Debug information support for variables in LDS

- ROCm 5.7 (August 2023)
  - Watchpoint on scratch memory,
  - Upstream KFD debugging interface and
  - Navi 3X support.

- ROCm 6.0 (November 2023)
  - MI 300 support

Debugger: DWARF GPU Extension Proposal Chronicle

- Since our discussion with **Ben Woodard** one year ago when DWARF extension proposal was in the stall....

- Formed a new workgroup called “DWARF extensions for GPUs,” which includes multiple members of the DWARF committee as well as representatives from the relevant companies
  - Perforce, Intel, Red Hat, Mentor Graphics, Cary Coutant
- The workgroup ended up splitting the original proposal into a series of smaller proposals, where each of the proposals are localized to a specific part of the DWARF specification
- Cary Coutant was appointed as a new DWARF committee chairman after previous one (Michael Eager) retired.
- Cary still felt that he could make an easier to digest "Location Descriptions on the Stack" proposal, which would also include the new offset composition operations
- Normative text changes, expression evaluation context definition and other needed extensions, are all planned to be submitted later and are still being discussed within the workgroup.
- The "Location Descriptions on the Stack" proposal was added to the DWARF issue queue on 25th of May and is currently being discussed within the DWARF committee

Debugger: Upstreaming Effort

- Basic support for AMDGPU debugging (enough to set a breakpoint in GPU code and run to it) https://gitlab.com/gnutools/binutils-gdb/-/commit/18b4d0736bc570c6d2e3e5f6ebc2ad4617d93847
- Fork and exec handling https://gitlab.com/gnutools/binutils-gdb/-/commit/5f6d638d3cb8273dac5c5bc1b541066dc41c7bb1

- Planned
  - Precise memory support
  - DWARF expression evaluator reimplementation (pending DWARF proposal progress)
  - Address space support (pending DWARF proposal progress)

Debugger: Compiler Support

- Added support for generating debug information for LDS variables
- Added support for architecture address spaces as well as language memory spaces to the generated type debug information
- Working closely with LLVM upstream community in upstreaming our debug info LLVM IR extensions

ROCm Debugger and API Useful Links

- AMD Debugger ROCgdb https://github.com/ROCm-Developer-Tools/ROCgdb
- AMD Debugger API (ROCdbgapi) https://github.com/ROCm-Developer-Tools/ROCdbgapi
- ROCgdb Documentation https://rocm.docs.amd.com/projects/ROCgdb/en/latest/index.html
- ROCdbgapi Documentation https://rocmdocs.amd.com/projects/ROCdbgapi/en/latest/

OMPT Target Implementation Status

OMPT Target Implementation Status in ROCm

- Target callbacks invoked at OpenMP target runtime entry points
  - Provide tools a way to establish an anchor, e.g., stack frame
- Trace records returned asynchronously to the tool
  - Contain information (e.g., timing) corresponding to the anchors
- Target callbacks/trace records implemented for all mandatory execution events
  - Target region, data operations, kernel submit
  - Implementation is part of the OpenMP runtime library and the AMDGPU plugin
- Available starting from ROCm 5.1

OMPT Tool Integration Status

- OMPT target support integrated with HPCToolkit and TAU
- Kernel execution and data transfer timings and metadata can be seen in callgraph profiles and execution traces
- Score-P integration in progress
- Upstreaming to LLVM trunk in progress, some patches landed, others under review

hpcviewer @hpe-sjc2-37

File View Filter Help

Profile: veccopy | Trace: veccopy

veccopy.c

```
15
16  for (i=0; i<N; i++)
17    b[i]=i;
18
19 #pragma omp target parallel for
20  {
21    for (int j = 0; j< N; j++)
22      a[j]=b[j];
23  }
24
25 #pragma omp target teams distribute parallel for
26  {
27    for (int j = 0; j< N; j++)
28
```

Top-down view | Bottom-up view | Flat view

Scope	CPUTIME (sec): Sum (I)	CPUTIME (sec): Sum (E)	GPUOP (sec): Sum (I)	GPUOP (sec): Sum (E)	GKER (sec): Sum (I)	GKER (sec): Sum (E)	GMEM (sec): Sum (I)	GME
« <omp tgt kernel>			2.07e-04 49.7%	2.07e-04 49.7%	2.07e-04 100.0%	2.07e-04 100.0%		
« OmptInterface::target_begin(long, void*) ...			2.07e-04 49.7%	2.07e-04 49.7%	2.07e-04 100.0%	2.07e-04 100.0%		
« __tgt_target_kernel [libomptarget.so.17...			2.07e-04 49.7%	2.07e-04 49.7%	2.07e-04 100.0%	2.07e-04 100.0%		
19 « main			1.90e-04 45.6%	1.90e-04 45.6%	1.90e-04 91.7%	1.90e-04 91.7%		
« <program root>			1.90e-04 45.6%	1.90e-04 45.6%	1.90e-04 91.7%	1.90e-04 91.7%		
25 « main			1.71e-05 4.1%	1.71e-05 4.1%	1.71e-05 8.3%	1.71e-05 8.3%		
<gpu copyin>			8.94e-05 21.4%	8.94e-05 21.4%				
« <omp tgt copyin>			8.94e-05 21.4%	8.94e-05 21.4%				
« OmptInterface::target_begin(long, void*) ...			8.94e-05 21.4%	8.94e-05 21.4%				
« __tgt_target_kernel [libomptarget.so.17...			8.94e-05 21.4%	8.94e-05 21.4%				
25 « main			4.51e-05 10.8%	4.51e-05 10.8%				
« <program root>			4.51e-05 10.8%	4.51e-05 10.8%				
19 « main			4.43e-05 10.6%	4.43e-05 10.6%				
<gpu copyout>			8.10e-05 19.4%	8.10e-05 19.4%				
<gpu alloc>			2.99e-05 7.2%	2.99e-05 7.2%			2.99e-05 75.4%	

Binary Instrumentation Support:
AMD GPU Machine Readable XML ISA Specification

Project Objectives

Joint effort with AMD RTG Tools and GPU Architecture teams

Develop AMD GPU shader ISA specification with following features:

- **Auto-generated** from the internal hardware definition files to be consistent with all details and product versions
- **Machine-Readable (M-R) XML format** to be used by binary instrumentation tools and similators (internal and 3rd party)
- **Executable**, i.e., provides essential information about execution semantics of the instructions
- **Set of auxiliary tools (API)** to conveniently access the information in XML specification

Related Works

- ARM specified their CPU architecture in XML
  - https://alastairreid.github.io/ARM-v8a-xml-release/
- IBM's Genesys-Pro instruction generator uses **architectural descriptions** as an input in machine readable form
  - https://uobdv.github.io/Design-Verification/Supplementary/GenesysPro.pdf
- Commercial verification tool RAVEN by Obsidian Software uses machine readable architecture specifications for PowerPC
  - https://www.slideshare.net/DVClub/introducing-obsidian-software-and-ravengcs-for-powerpc
- Academic work by Kamkin et al. developed a machine-readable specification for RISC-V and ARM architectures in nML language
  - https://riscv.org/wp-content/uploads/2018/12/Machine-Readable-Specifications-of-RISC-V-ISA-Kamkin-Tatarnikov.pdf
  - https://ieeexplore.ieee.org/document/8746054
- To the best of our knowledge, **no similar work has been done in GPU domain**

XML ISA Info Layout

- **Encodings**:
  - Contains information about all encodings supported by this architecture
  - Examples of the provided information by this element: instruction sizes, fields of the binary instruction, general description of the encoding
- **Instructions**:
  - List of all instructions in particular GPU architecture
    - This is the core element of the specification
    - Every instruction references other XML elements
    - Examples of provided information by this element: different ways to encode the instruction, the type of operands, the data format of the operand, etc.
- **Data Formats**:
  - Provides additional information on how the values in the registers should be treated
    - This element is referenced by instruction element.
    - Examples of provided information by this element: is the value integer or float? If it is float where is mantissa, exponent and sign?
- **Operand Types:**
  - The sub-elements of this element are referenced by an instruction element
  - It provides information on the types of the operands used by the instruction
    - Examples of provided information are: is the operand a scalar or a vector register? What is the name of this operand when represented in assembly?

Current Version of the Tool

- The XML ISA spec is released with the utility API that provides interfaces for the common use cases, such as decoding of the binary instructions and kernels/shaders
- The beta version of the tool has been shared with internal teams and let to productivity improvements in the development of tools that require ISA data
- It has been also shared with AMD partners and business clients
  - We constantly receive feedback and try to improve the tool

Ongoing Active Development

- The tools is getting polished for public release
- There is an ongoing effort to embed semantics information into the XML specification
- Methodology (the bird’s-eye view)
  - Leveraging existing “pseudocode” in the conventional ISA documentation
  - The pseudocode is parsed and represented in a form of an abstract syntax tree (AST)
  - The AST is embedded into the XML
- Challenges
  - The pseudocode needs to be type checked and functionally verified

ROCprof, Advanced Thread Trace, OmniTools and uProf

Brief update on suite of AMD performance analysis tools

- Two versions of **ROCprof** tool provided
  - **ROCprof V1** tool uses ROCprofiler V1 API
  - **ROCprof V2** tool uses ROCprofiler V2 API
  - The functionality of ROCprof V2 tool is a superset of ROCprof V1 tool
    - ROCprof V2 tool has greater support for plugins
- **OmniPerf** and **OmniTrace** tools
  - Initially may be delivered as separate packages
  - Omnitrace has some initial support for multi-node
- **Advanced Thread Trace (ATT)** tool
  - Delivered in ROCm 5.5 with a web interface
  - Planning to implement a stand-alone video interface
- Continue to support 3rd party tools through ROCprofiler API and ROCprof tool plugins

AMD Research OmniTrace and OmniPerf AMDGPU performance model document

- **Omnitrace** is a comprehensive profiling and tracing tool for parallel applications written in C, C++, Fortran, HIP, OpenCL, and Python which execute on the CPU or CPU+GPU
  - OmniTrace Page https://github.com/AMDResearch/omnitrace
- **OmniPerf** is a system performance profiling tool for machine learning/HPC workloads running on AMD MI GPUs. The tool presently targets usage on MI100 and MI200 accelerators.
  - **OmniPerf** Page https://github.com/AMDResearch/omniperf

- **OmniPerf** AMDGPU performance model document
  - OmniPerf AMDGPU performance model document was developed
  - Shared under NDA at the moment, final version will go to public domain

ROCm Tools: ROCm Profiler Saga V1 → V2

Some V1 Issues addressed in V2 Development

- Correctness issues
  - Correct oddities in API semantics (silently ignoring requests, …)
  - Issues with concurrency
- Issues with memory leaks
- Issues with scaling

- Implement as a single unified library versus two separate libs
- Clearly documented intuitive API much easier to follow

V2 versus V1 Delta

- Unified correlation ID across API operations and asynchronous operations
- Allow multiple services to be enabled at same time
- Allow multiple clients
- Client tool management
  - Provide means for client tools to be initialized cleanly
  - Helper thread creation notification
- Kernel dispatch serialization for profile counter collection
- Plugin interface to support multiple output trace formats (JSON, CTF, OTF-2, etc.)
- Addresses performance issues with post processing in V1

Changes in V2 since first release

- V2 is released as a beta interface that is being evolved in response to user feedback
- User feedback is driving the changes and makes them acceptable via info about identified API issues
- Examples
  - Fixed header files so can include both tracer and profiler
  - Removed C++'isms
  - Fixed ROCtracer deadlock issue
  - Fixed other issues early adopters reported

  - It is ongoing process until V2 architecture will be hardened (based on user's feedback)

Feedback on V2 received from tool developers (ROCprof and 3rd party)

- ROCm ROCprof V2 tool is using the ROCprofiler V2 API
- Some of the 3rd party tools have investigated the current ROCprofiler V2 API
- Major feedback is that there are numerous issues that need to be resolved before V2 can be used properly
  - Limitation of single session not compatible with multiple clients
  - Missing mapping of operation codes to operation names
  - Missing calling on entry and exit for API tracing
  - Not possible to enable buffered tracing for APIs
  - Memory and performance issues
- Many of these issues are being fixed in upcoming releases
- Other issues being addressed by changes to API planned for future releases
- Expect to allow the API evolving from user feedback and get stabilized once users satisfied

ROCprofiler V2 API

Disclaimer

- Following slides present work in progress on the evolution of the ROCprofiler V2 API
- The final design may differ as it is collaboratively evolved with the users of the API

Key Concepts

- **Clients**
  - Allows multiple tools to use the API concurrently
  - Initial restrictions on services that can be used by clients concurrently
- **Services**
  - Separate API for each functionality provided (callback tracing, kernel dispatch profiling, …)
- **Buffers**
  - Used by services that want efficient storing of multiple records
  - Separate thread to asynchronously return block of records when buffer reaches a certain size

Clients

- Create and destroy client handles
- All service operations take a client handle
- Initially only support multiple clients with disjoint services
  - Potential future support for multiple clients (tracing, markers…)

Services: Callback Tracing

- Can enable and disable operations independently
- Calls a client provided callback by same thread that invoked the operation
- HIP/HSA API
  - Public API functions
  - Provide full arguments of operation
  - Call on entry and exit
  - Includes host function registration
  - Creates correlation ID that is reported by non-API callback operations and buffed tracing that causes them
- Marker
  - Formerly **ROCtx**
- Code Object Tracing
  - Load
  - Unload
  - Kernel symbol enumeration
- Helper Thread Creation
  - Report creation of profiler and runtime helper threads so client tools can ignore them

Services: Buffered Tracing

- Can enable and disable operations independently
- Add “activity” record to a buffer when enabled operation is executed
  - Separate thread to asynchronously return block of records when buffer reaches a certain size
- Record has limited arguments of the operation
- For API operations creates correlation ID that is reported by non-API callback operations and buffer tracing that causes them
- Can enable callback and buffered tracing for operations they both support
- External correlation ID
- Buffered tracing
  - HIP/HSA API Tracing (only essential data put in record)
  - Marker
  - Memory copy
  - Kernel dispatch
  - Page migration
  - Scratch memory
- Correlation IDs
  - Connects API operations to asynchronous completion reporting

Services: Performance Counter Profiling

- Queries to determine counters available on each agent
  - Indicate if counter has multiple instances
  - Indicate result type of counter
  - Provide name of counter
  - Provide raw hardware counters
  - Provide derived metric counters defined by XML file
- Define sets of profile counters to collect
  - Can query how to collect a set of counters in multiple passes
- Kernel dispatch profiling
  - Invoke user callback when dispatch enqueued
    - Callback can decide if want to profile and return a profile counter set
  - Serializes kernel dispatches
- Agent profiling
  - Can enable a profile counter set
  - Can request counter values on demand
  - Allows device wide profiling

Services: Other

- Advanced Thread Tracing (ATT)
  - Available in V2 current release
- MI200 Page Migration Profiling
  - To be added
- PC sampling
  - In development
- Streaming Performance Monitors (SPM)
  - Not yet supported

Services: MI200 page migration profiling

- KFD provides ioctl to obtain a file handle used to report page migration events
- Can specify which events want reported
- Each event writes one fixed format text record to the file handle
- There is an example using it in ROCr (see SvmProfileControl)
- Planning to add page migration service to ROCprof V2 API to report these events

Services: PC Sampling

- Based on trap handler
  - Avoids requiring root privileges
- Support host call for MI200
  - Host call allows PC/Queue/Dispatch/CU information to be collected
- Support host call and hardware stochastic sampling in MI300
  - Stochastic sampling provides more detailed information about CU functional unit state
  - Same API and buffering for both MI200 and MI300
- Design of PC sampling progressing well
  - Defined KFD ioctl API
  - Defined ROCr runtime API
  - Designing trap handler and buffering algorithm
  - Working on efficient way to capture dispatch ID
    - Dispatch ID requested by HPCToolkit as it would allow to attribute PC samples to specific dispatches

Buffers

- Used by any service that wants to provided buffered output
- User control of:
  - buffer size
  - buffer limit to trigger asynchronous thread to use user callback to provide records
- Client can flush buffer on demand
- Considering mode to deliver records without buffering
  - Allows asynchronous operations to be reported synchronously when they happen

Client tool management

- Client can define a symbol that will be invoked when ROCprofiler API is first used
  - Avoids problems of initialization/finalization racing with global constructors
- Kernel serialization
  - not cross process

Kernel dispatch serialization

- API responsible for ensuring kernel dispatches are executed serially when necessary
  - Kernel dispatch profile counting needs serialization to get meaningful results
  - Need to determine how to handle multi process sharing of single device

Copyright and disclaimer

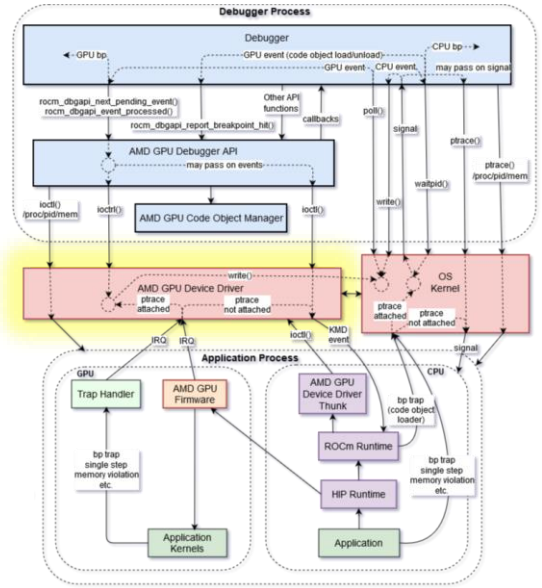
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.


AMD

BACKUP

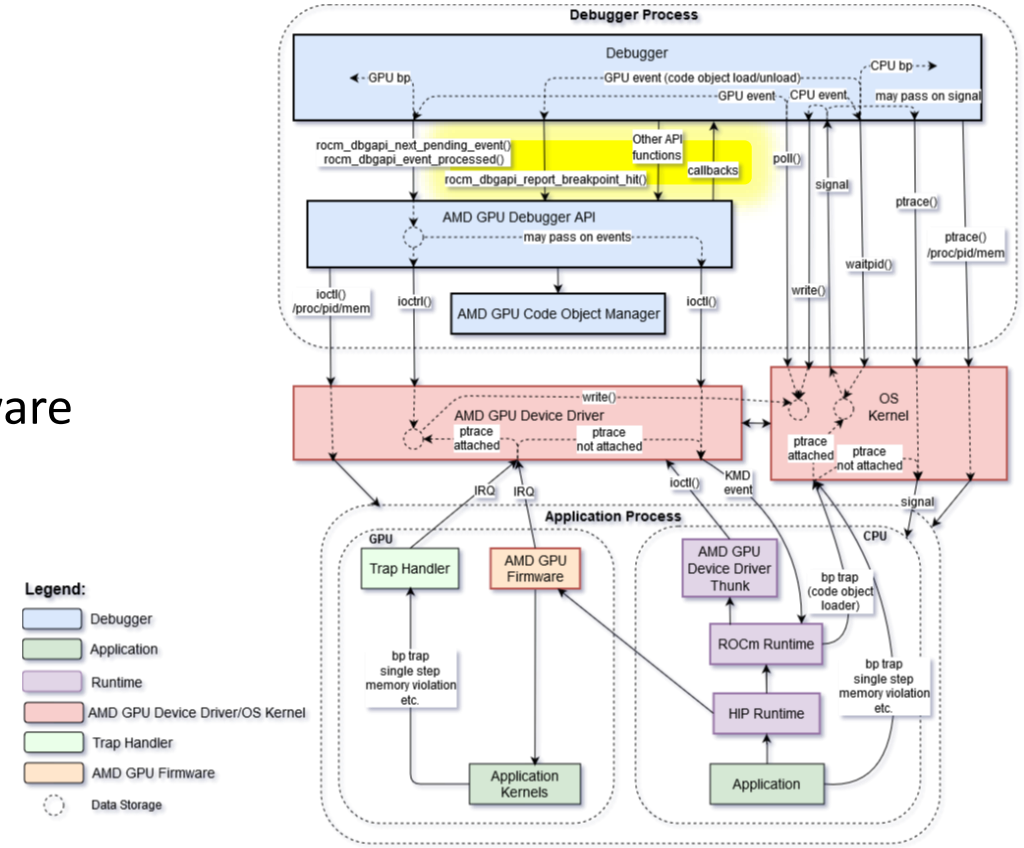
AMD GPU Device Driver (KFD)



- Model after CPU ptrace/waitpid
- Use a single ioctl with debug functions to control AMD GPU
- **A lot of effort in API design to avoid ABA and async races**
  - **Snapshot of queues**
    - Transactional to avoid data races
  - **New queue concept to avoid ABA issues**
    - Snapshot information indicates if queue has pending events, queue information
  - **Use of file descriptor per device to communicate events: wave interrupts cause queue events**
  - Next pending device event: similar to `waitpid` for process signals
  - Batched suspend/resume queues: Indicate which queues updated, which no longer exist (new queue concept)
  - Redirection of device events when debugger attached *(planned)*
    - Similar to CPU process signals being redirected to debugger by `ptrace`
- Currently based on CWSR
  - Added standard header to context save area so debugger can obtain necessary information
- First level trap handler support CWSR
  - Can invoke second level handler installed by ROCm low level runtime

AMD Debugger API (ROCdbgapi): Rationale

- **A library to control and query the AMD GPU hardware**
- Desire to abstract the specific idiosyncrasies of the hardware
  - Many contortions internally to present a simple clean view
- Allows the same API to support all current and future GPU hardware
- **Simple intuitive operations and concepts**
  - **Very similar in style to using CPU `ptrace` and `waitpid`**
  - **Use file descriptor and `poll` to report asynchronous events**
  - **Makes integrating with a debugger and other tools simpler**
- **Hides implementation details**
  - **Currently uses CWSR**
  - **Could use host trap for live wave debugging in the future with no change of interface**
    - Necessary if extended to support graphics which does not have CWSR
- Being used by multiple tools
  - ROCgdb, Perforce TotalView debugger, ROCm Debug Agent
- Full Doxygen documentation



ROCm Low Level Runtime (ROCr)

- **Deliberately made dependence on the runtime as minimal as possible**
  - **Only code object loading is needed**
  - **All other information comes directly from kernel driver**
  - Avoids any other interfaces to runtime
  - Makes supporting different runtimes easier
- Second level trap handler
  - Entered due to: `S_TRAP`, single step, memory violation, address watch
  - Halts wave and records reason
  - Interrupts kernel driver which reports a queue event
- **Supports code object loading**
  - **Model after Linux dynamic loader:**
    - `r_debug` structure:
      - Linked list of loaded code objects
      - State variable to indicate if list being updated
    - `r_break` function called before start/end update
      - Debugger can set breakpoint to prevent updates to list while reading it