# ROCgdb, GDB and AMDGPU debugging

**Lancelot SIX**

AMD

together we advance_

[Public]

# Agenda

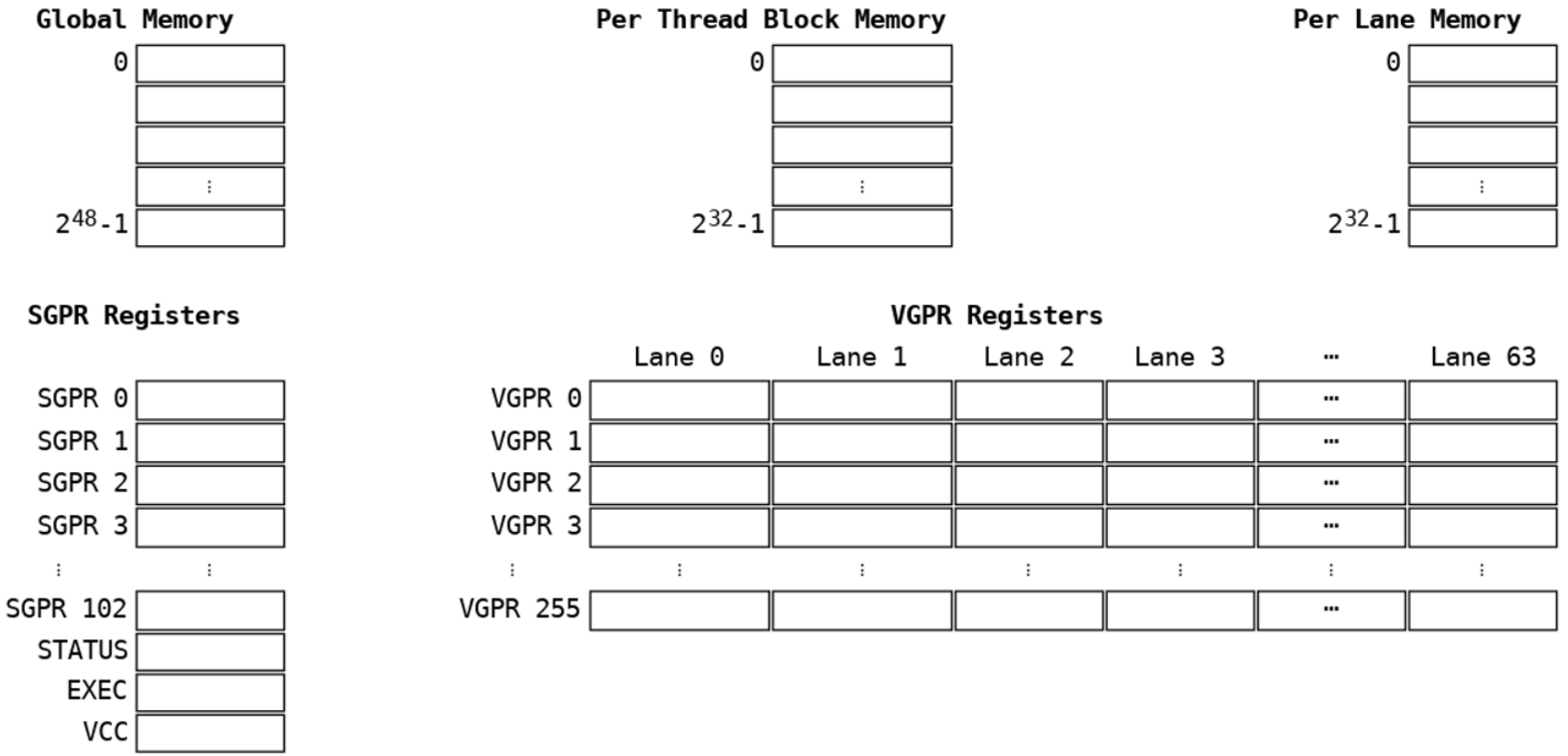
- AMDGPU architecture overview and programming model.
- ROCgdb.
- State of AMDGPU support in upstream GDB.



AMD
together we advance_

[Public]

# Abstract AMDGPU architecture




- Multiple processing elements form a compute unit.
- Multiple compute units form a GPU.



AMD
together we advance_

[Public]

# The HIP "hello world"

```
#include <hip/hip_runtime.h>

__global__ void
kern (int *arr) {
  arr[blockIdx.x * blockDim.x + threadIdx.x] += 2;
}


int
main () {
  const size_t block_size = 64, blocks = 16,
               size = block_size * blocks;
  std::array<int, size> data = {};


  int *device_ptr;
  hipMalloc (&device_ptr, size * sizeof (int));
  hipMemcpyHtoD (device_ptr, data.data (), size * sizeof (int));

  kern<<<blocks, block_size>>> (device_ptr);


  hipMemcpyDtoH (data.data (), device_ptr, size * sizeof (int));
  return EXIT_SUCCESS;
}
```

GPU/device code

CPU/host code



AMD
together we advance_

[Public]

# Host threads and GPU threads (waves) under single inferior

- Waves are mapped to threads in GDB.
- Same program with unified memory.

```
(gdb) info threads
  Id   Target Id                                         Frame
  1    Thread 0x7ffff7e69a80 (LWP 152926) "a.out" 0x…f41 in ?? () from …/libhsa-runtime64.so.1
  2    Thread 0x7fffeb3ff640 (LWP 152929) "a.out" __GI___ioctl (fd=3, request=3222817548) …
  5    Thread 0x7ffff62d7640 (LWP 152933) "a.out" __GI___ioctl (fd=3, request=3222817548) …
* 6    AMDGPU Wave 1:1:1:1 (0,0,0)/0 "a.out"      kern (arr=0x7fffe9c00000) at simple.cpp:5
  7    AMDGPU Wave 1:1:1:2 (0,0,0)/1 "a.out"      kern (arr=0x7fffe9c00000) at simple.cpp:5
  8    AMDGPU Wave 1:1:1:3 (1,0,0)/0 "a.out"      kern (arr=0x7fffe9c00000) at simple.cpp:5
  …
```



AMD
together we advance_

[Public]

# GPU Threads (waves)'s Target Id

```
(gdb) info threads
…
                   /- agent
                   | /- queue
                   | | /- dispatch
                   | | | /- wave id
                   | | | |
  6    AMDGPU Wave 1:1:1:1 (0,0,0)/0 "a.out"      kern (arr=0x7fffe9c00000) at simple.cpp:5
                             ^^^^^  |
                                |    \- wave number in work group
                                \- work group coordinates in work grid
…
```

- "info {agent, queue, dispatch}" commands also available.



AMD
together we advance_

[Public]

# Lanes

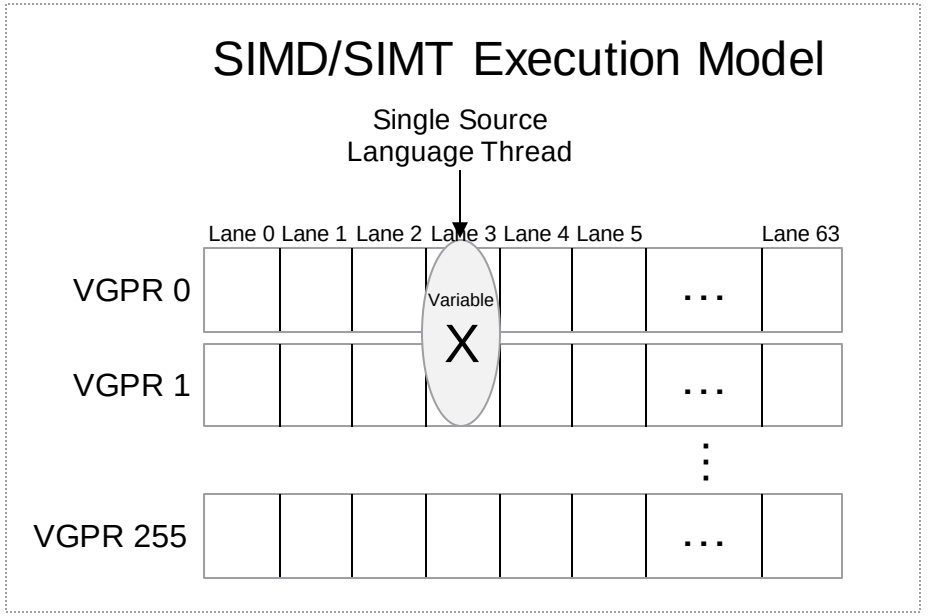
New entity under threads: threads become vectorized, multiple lanes under one thread.

GDB threads are mapped to GPU waves. All lanes progress side-by-side forming a wavefront.

One physical PC for the whole thread (for all lanes), but:

- Each lane works with its own slice of the register set, on its share of data, its version of locals in scope.
- Lanes can be seen as multiple "regular" threads running in lockstep.

"current lane" concept added (augmenting "current inferior", "current thread").






AMD
together we advance_

[Public]

# Lanes commands

```
(gdb) info lanes
  Id   State Target Id                                    Frame
* 0    A     AMDGPU Lane 1:1:1:1/0 (0,0,0)[0,0,0]   kern (arr=0x7fffe9c00000) …
  1    A     AMDGPU Lane 1:1:1:1/1 (0,0,0)[1,0,0]   kern (arr=0x7fffe9c00000) …
  2    A     AMDGPU Lane 1:1:1:1/2 (0,0,0)[2,0,0]   kern (arr=0x7fffe9c00000) …
  3    A     AMDGPU Lane 1:1:1:1/3 (0,0,0)[3,0,0]   kern (arr=0x7fffe9c00000) …
  4    A     AMDGPU Lane 1:1:1:1/4 (0,0,0)[4,0,0]   kern (arr=0x7fffe9c00000) …
  5    A     AMDGPU Lane 1:1:1:1/5 (0,0,0)[5,0,0]   kern (arr=0x7fffe9c00000) …
  6    A     AMDGPU Lane 1:1:1:1/6 (0,0,0)[6,0,0]   kern (arr=0x7fffe9c00000) …
…
  31   A     AMDGPU Lane 1:1:1:1/31 (0,0,0)[31,0,0] kern (arr=0x7fffe9c00000) …
(gdb) lane 8
[Switching to thread 6, lane 8 (AMDGPU Lane 1:1:1:1/8 (0,0,0)[8,0,0])]
#0  kern (arr=0x7fffe9c00000) at simple.cpp:5
5         arr[blockIdx.x * blockDim.x + threadIdx.x) += 2;
```



AMD
together we advance_

[Public]

# SIMT Lanes's Target Id

```
                  /- agent
                  | /- queue
                  | | /- dispatch
                  | | | /- wave id
                  | | | |
                  | | | |
AMDGPU Lane 1:1:1:1/8 (0,0,0)[8,0,0]   kern (arr=0x7fffe9c00000) at simple.cpp:5
                        |  ^^^^^  ^^^^^
                        |    |      |
                        |    |      \- work item coordinates in work group
                        |    \- work group coordinates in work grid
                        \- lane index
```



AMD
together we advance_

[Public]

# SIMT and Lane divergence


```
if (foo (lid)) {
  elem = in[lid] + 1;
} else {
  elem = in[lid] + 3;
}
```

Else lanes

Then lanes


```
// if (foo (lid)) {
     NoOp;
// } else {
     elem = in[lid] + 3;
// }
```


```
// if (foo (lid)) {
     elem = in[lid] + 1;
// } else {
     NoOp;
// }
```



AMD
together we advance_

[Public]

# Without lane divergence support, step 1

**Stepping stops in all branches => surprising**

```
        __device__ void
        function (unsigned tid, const int *in, int *out)
        {
          int elem;

>> (1)    if (tid % 2)                    <<<<<<<<<
   (3)      elem = in[tid] + 1;
          else
   (2)      elem = in[tid] + 3;
   (4)    atomicAdd (out, elem);
        }
```



AMD
together we advance_

[Public]

# Without lane divergence support, step 2

**Stepping stops in all branches => surprising**

```
        __device__ void
        function (unsigned tid, const int *in, int *out)
        {
          int elem;

   (1)    if (tid % 2)
   (3)      elem = in[tid] + 1;
          else
>> (2)      elem = in[tid] + 3;        <<<<<<<<<
   (4)    atomicAdd (out, elem);
        }
```



AMD
together we advance_

[Public]

# Without lane divergence support, step 3

**Stepping stops in all branches => surprising**

```
        __device__ void
        function (unsigned tid, const int *in, int *out)
        {
          int elem;

   (1)    if (tid % 2)
>> (3)      elem = in[tid] + 1;        <<<<<<<<<
          else
   (2)      elem = in[tid] + 3;
   (4)    atomicAdd (out, elem);
        }
```



AMD
together we advance_

[Public]

# Without lane divergence support, step 4

**Stepping stops in all branches => surprising**

```
        __device__ void
        function (unsigned tid, const int *in, int *out)
        {
          int elem;

   (1)    if (tid % 2)
   (3)      elem = in[tid] + 1;
          else
   (2)      elem = in[tid] + 3;
>> (4)    atomicAdd (out, elem);       <<<<<<<<<
        }
```



AMD
together we advance_

[Public]

# With lane divergence support, step 1

**Stepping doesn't stop if current lane is inactive => intuitive**

```
        __device__ void
        function (unsigned tid, const int *in, int *out)
        {
          int elem;

>> (1)    if (tid % 2)                    <<<<<<<<<
   (X)      elem = in[tid] + 1;
          else
   (2)      elem = in[tid] + 3;
   (3)    atomicAdd (out, elem);
        }
```



AMD
together we advance_

[Public]

# With lane divergence support, step 2

**Stepping doesn't stop if current lane is inactive => intuitive**

```
        __device__ void
        function (unsigned tid, const int *in, int *out)
        {
          int elem;

   (1)    if (tid % 2)
   (X)      elem = in[tid] + 1;
          else
>> (2)      elem = in[tid] + 3;          <<<<<<<<<
   (3)    atomicAdd (out, elem);
        }
```



AMD
together we advance_

[Public]

# With lane divergence support, step 3

**Stepping doesn't stop if current lane is inactive => intuitive**

```
        __device__ void
        function (unsigned tid, const int *in, int *out)
        {
          int elem;

   (1)    if (tid % 2)
   (X)      elem = in[tid] + 1;
          else
   (2)      elem = in[tid] + 3;
>> (3)    atomicAdd (out, elem);       <<<<<<<<<
        }
```



AMD
together we advance_

[Public]

# Multiple address spaces support

- Introducing the `address_space#offset` notation.
- Architecture address-spaces are not a source language concept!
- Use "`maintenance print address-spaces` " to list the address-spaces supported by the current target.

```
__device__ int some_global = 8;
__shared__ int some_shared;

__device__ int
func (int *arr)
{
  int some_private = 8;
  return some_shared + some_private;
}
```

```
(gdb) p &some_global
$1 = (int *) 0x7ffff617d1a8 <some_global>
(gdb) p &some_shared
$2 = (int *) local#0x0
(gdb) p &some_private
$3 = (int *) private_lane#0x40
(gdb) x private_lane#0x40
private_lane#0x40:      0x00000008
(gdb) p *(int*)private_lane#0x40
$4 = 8
```



AMD
together we advance_

[Public]

# Useful things to know

- Exceptions are reported for the entire wave.

- Memory exceptions are not precise by default.
  - If available, use "`set amdgpu precise-memory on`" to identify the faulty instruction (impacts performances).

- On some architectures, attaching to a process shows:

```
AMDGPU Wave 1:3:?:1 (?,?,?)/? "attached_process" …
```

  - Use "HSA_ENABLE_DEBUG=1" before starting the process.

- On some architectures, at most one process can be debugged at a time.



AMD
together we advance_

[Public]

# DWARF and GPU architectures

- Challenges with current DWARF standard:
  - AMD GPUs have multiple address spaces, a "numerical address" is not sufficient to locate an object in memory.
  - The stack is not implemented in the default address space.
  - Support for divergent control flow of SIMT hardware.
  - And more…
- DWARF extensions to overcome those issues implemented in downstream LLVM + ROCgdb:
  - https://llvm.org/docs/AMDGPUDwarfExtensionsForHeterogeneousDebugging.html
- "DWARF for GPU" workgroup involving multiple vendors working to propose changes to the DWARF committee:
  - https://dwarfstd.org/issues/230524.1.html
  - More to come…



AMD
together we advance_

[Public]

# AMDGPU support status in upstream GDB

| Feature | Status |
|---|---|
| ~~Driver support in upstream Linux kernel~~ | ✓ |
| ~~Wave control (breakpoint, single-step, interrupt & resume)~~ | ✓ |
| ~~Disassembly~~ | ✓ |
| ~~Precise memory operations~~ | ✓ |
| Displaced stepping | Pending |
| Stack unwinding | Blocked by DWARF |
| Printing variables | Blocked by DWARF |
| Address spaces support | Blocked by DWARF |
| Lane debugging | Need to agree with other vendors on the UI.<br>Need DWARF to write testcases. |
| Watchpoints | Need DWARF to write testcases |



AMD
together we advance_

[Public]

# Additional resources

- ROCgdb:
  - https://github.com/ROCm/ROCgdb/
  - https://rocm.docs.amd.com/projects/ROCgdb/en/latest/ROCgdb/gdb/doc/gdb/Heterogeneous-Debugging.html
  - https://rocm.docs.amd.com/projects/ROCgdb/en/latest/ROCgdb/gdb/doc/gdb/AMD-GPU.html
- Rocm-dbgapi (used by GDB to control AMDGPU devices):
  - https://github.com/ROCm/ROCdbgapi/
- "Anatomy of ROCgdb" at GNU Cauldron 2022:
  - https://gcc.gnu.org/wiki/cauldron2022#cauldron2022talks.anatomy_of_rocgdb_gdb_for_amd_gpus
  - https://www.youtube.com/watch?v=X1iZ_Ta7jOo
- "DWARF extensions for optimized SIMT/SIMD (GPU) debugging" at Linux Plumbers Conference 2021:
  - https://lpc.events/event/11/contributions/1012/
  - https://www.youtube.com/watch?v=Iv2WO67nklc



AMD
together we advance_

[Public]

# Disclaimers and Attributions

The information contained herein is for informational purposes only, and is subject to change without notice. Timelines, roadmaps, and/or product release dates shown in these slides are plans only and subject to change.

While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

©2024 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, EPYC, ROCm, CDNA, RDNA and combinations thereof are trademarks of Advanced

Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.



AMD
together we advance_


AMD