HIP Training Workshop – Day 3

ROCgdb & HIP math libraries

Justin Chang
Part 1: Intro to ROCgdb
Getting started

What is ROCgdb, from the tin:

The ROCm Debugger (ROCgdb) is the ROCm source-level debugger for Linux, based on the GNU Debugger (GDB). It enables heterogenous debugging on the ROCm platform of an x86-based host architecture along with AMD GPU architectures supported by the AMD Debugger API Library (ROCdbgapi). The AMD Debugger API Library (ROCdbgapi) is included with the ROCm release.

The current ROCm Debugger (ROCgdb) is an initial prototype that focuses on source line debugging and does not provide symbolic variable debugging capabilities. The user guide presents features and commands that may be implemented in future versions.

So... cuda-gdb? Yes, and mostly no -- rocgdb is (or will be) gdb, that is it tracks upstream GDB master.
What can it do?

In addition to your usual host-debugging capabilities, a very brief overview of rocgdb's current functionality:

- Switching between and seeing info about wavefronts
- Read/write to hardware registers, global memory, and LDS/scratch
- Breakpoints
- Watchpoints
- ISA-level debugging, mapping of ISA to source lines

Before we can take a test drive, let's talk about the code we're going to use it on.
Jacobi Example

Our example code:

- Solves a Laplacian problem
- 5-point finite difference stencil
- May be distributed over multiple MPI ranks
- Performs several iteration of a Jacobi method
Laplacian equation

Laplace equation:

\[-\nabla^2 q = f\]

In two dimensions, with a 5-point finite difference stencil we discretize as:

\[\frac{-q_{i-1,j} + 2q_{i,j} - q_{i+1,j}}{\Delta x^2} + \frac{-q_{i,j-1} + 2q_{i,j} - q_{i,j+1}}{\Delta y^2} = f_{i,j}\]

On a rectangular lattice one point in the x–y plane:
Jacobi Example

- The finite difference Laplacian operator is a sparse matrix operator on a vector of unknowns:

\[ Aq = f \]

- When the whole domain is distributed with MPI, each process will need to receive halo data to evaluate \( Aq \)

- Can compute Laplacian at interior stencil points while the halo data is being exchanged

- Jacobi Iterative method is the iteration:

\[ q^{k+1} = q^k + D^{-1}(f - Aq^k) \]

where \( D \) is the diagonal of \( A \).
Jacobi Structure

- Initialize MPI
- Setup *Jacobi* object
  - ApplyTopology()
  - CreateMesh()
  - InitializeData()
- Execute *Jacobi* run method
Jacobi Structure

- Initialize MPI
- Setup *Jacobi* object
  - ApplyTopology()
    - Establish MPI neighbors
    - Set GPU devices
  - CreateMesh()
  - InitializeData()
- Execute *Jacobi* run method

```c
//select GPU
int devCount = 0;
SafeHipCall(hipGetDeviceCount(&devCount));

if (devCount == 0) {
    fprintf(stderr, "Error: MPI rank %d detected no GPUs."
             "Terminating...\n", grid.rank);
    MPI_Abort(grid.comm, STATUS_ERR);
}

char name[255];
gethostname(name, 255);
long int hostId = gethostid();
long int* hostIds = (long int*) calloc(size,sizeof(long int));
MPI_Allgather(&hostId,1,MPI_LONG,hostIds,1,MPI_LONG,grid.comm);

//count how many ranks are on this node
int local_rank = 0;
for (int r=0;r<grid.rank;r++)
    if (hostIds[r]==hostId) local_rank++;

//select gpu via round-robin
grid.device_id = local_rank % devCount;
SafeHipCall(hipSetDevice(grid.device_id));
```
Jacobi Structure

- Initialize MPI
- Setup *Jacobi* object
  - ApplyTopology()
    - Establish MPI neighbors
    - Set GPU devices
  - CreateMesh()
    - Set local domain
    - Create halo exchange buffers
    - Create streams
  - InitializeData()
- Execute *Jacobi* run method
Jacobi Structure

- Initialize MPI
- Setup *Jacobi* object
  - ApplyTopology()
    - Establish MPI neighbors
    - Set GPU devices
  - CreateMesh()
    - Set local domain
    - Create halo exchange buffers
    - Create streams
  - InitializeData()
    - Generate initial data
    - Copy data to device
- Execute *Jacobi* run method
Iterative Loop

```c
while(){
    LocalLaplacian()
    HaloExchange()
    HaloLaplacian()
    JacobiIteration()
    Norm()
}
```

- **Note:**
  - Two streams: computeStream and dataStream
  - hipEventRecord timings on each respective stream (omitted for clarity)
Preparing the code for the debugger

Preparing your HIP code for debugging:
• Use any optimization level you like, we'll use -O3
• Have ROCm load code objects at initialization:
  • export HIP_ENABLE_DEFERRED_LOADING=0
• Add `-ggdb` to your flags:
• Optionally print even more useful information on API calls
  • export AMD_LOG_LEVEL=3

Example of what the compile options may look like...

```bash
mpic++ -I/usr/lib/x86_64-linux-gnu/openmpi/include/openmpi \  -L/usr/lib/x86_64-linux-gnu/openmpi/include -pthread -O3 -g -ggdb -fPIC \  -std=c++11 \ -march=native -Wall -I/opt/rocm/roctracer/include \  -I"/opt/rocm-4.2.0/hip/include" -I"/opt/rocm/llvm/bin/../lib/clang/12.0.0" \  -I/opt/rocm/hsa/include -I/opt/rocm/roctracer/include \  -c JacobiSetup.cpp -o JacobiSetup.o
```
Diving in

Launching the debugger, is the same as gdb:

rocgdb --args ./Jacobi_hip -g 1 1

For this demo, I will be using a program called cgdb alongside rocgdb, this gives a nice curses-based interface, but is by no means required. In this case:

cgdb -d rocgdb --args ./Jacobi_hip -g 1 1
Setting a breakpoint

Let’s step into one of our kernels, LocalLaplacianKernel computes the finite difference for one of our MPI domains.
Setting a breakpoint in host code

Here we setup a breakpoint in the host code. We can inspect the device pointer and its values:
Setting a breakpoint in host code

Typing ‘step’ enables one to iterate through the stack trace:
Setting a breakpoint in device kernel

Invoke ‘b’ or ‘break’ to the device kernel of interest:
Setting a breakpoint in device kernel

What happens when you type ‘step’? Another thread hit the same breakpoint! GDB will switch context to the new thread:
Setting a breakpoint in device kernel

[Switching to AMDGPU Thread 1:5:1:1 (0,0,0)/0]

Thread 9 "Jacobi_hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized out>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optimized out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9

AMDGPU Thread agent-id:queue-id:dispatch-num:wave-id (work-group-z,work-group-y,work-group-x)/work-group-thread-index
Setting a breakpoint in device kernel

- `agent-id = Agent Target ID`

```
[Switching to AMDGPU Thread 1:5:1:1 (0,0,0)/0]
Thread 9 "Jacobi hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized out>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optimized out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9

AMDGPU Thread agent-id:queue-id:dispatch-num:wave-id (work-group-z,work-group-y,work-group-x)/work-group-thread-index
```
Setting a breakpoint in device kernel

- agent-id = Agent Target ID
- queue-id = Queue Target ID
Setting a breakpoint in device kernel

- agent-id = Agent Target ID
- queue-id = Queue Target ID
- dispatch-num = Dispatch Target ID – how many kernels have been launched

[Switching to AMDGPU Thread 1:5:1:1 (0,0,0)/0]

Thread 9 "Jacobi_hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized out>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optimized out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9

AMDGPU Thread agent-id:queue-id:dispatch-num:wave-id (work-group-z,work-group-y,work-group-x)/work-group-thread-index
Setting a breakpoint in device kernel

- **agent-id** = Agent Target ID
- **queue-id** = Queue Target ID
- **dispatch-num** = Dispatch Target ID – how many kernels have been launched
- **wave-id** = Wavefront ID – index of wavefront of kernel

```
[Switching to AMDGPU Thread 1:5:1:1 (0,0,0)/0]

Thread 9 "Jacobi_hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized out>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optimized out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9
```

AMDGPU Thread agent-id:queue-id:dispatch-num:wave-id (work-group-z,work-group-y,work-group-x)/work-group-thread-index
Setting a breakpoint in device kernel

- agent-id = Agent Target ID
- queue-id = Queue Target ID
- dispatch-num = Dispatch Target ID – how many kernels have been launched
- wave-id = Wavefront ID – index of wavefront of kernel
- \((z, y, x)\) work-group/block index

[Switching to AMDGPU Thread 1:5:1:1 (0,0,0)/0]

Thread 9 "Jacobi_hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized out>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optimized out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9

AMDGPU Thread agent-id:queue-id:dispatch-num:wave-id (work-group-z,work-group-y,work-group-x)/work-group-thread-index
To look at the progress of a single wavefront, disable the breakpoint by typing ‘disable <num>’, and then ‘step’
Setting a breakpoint in device kernel

Use ‘stepi’ which enables the user to dive deeper into the HIP API.
Setting a breakpoint in device kernel

Type ‘step’ again and you can now freely step through the device kernel
Examinining the ISA

Several ways one can view the ISA. Using cgdb, type

ESC -> :set dis -> ENTER
Let's say I want to see what the value of "i" is for my wavefront:
Examining the ISA

This also works for scalar registers, for instance here we can check the conditional:

\[(i < \text{localNx}) \&\& (j < \text{localNy})\]

which is stored in s[0:1], one bit corresponding to each thread in the wavefront:

```cpp
if (i < localNx) && (j < localNy) {  
  if (i < localNx) {  
    v_add_u32_e32 v0, v8, v0  
  }
```

```cpp
75 | const int id = (i+1) + (j*1); stride;  
76 | v_add_u32_e32 v1, v1, v1  
77 | v_mullo u32 v1, v1, v1  
78 | v_load_dwordx2 s[0:1], v1, s[0:1], 0x20
```

```cpp
9 | constexpr void LocalLaplacianKernel(const int localNx,  
9 |   const int localNy,  
9 |   float dx, float dy)  
9 |   {  
9 |     for (int i = 0; i < localNx; i++) {  
9 |       for (int j = 0; j < localNy; j++) {  
9 |         float u0 = u[i][j], u1 = u[i][j+1], u2 = u[i+1][j], u3 = u[i+1][j+1];  
9 |         float uu = u0 + u1 + u2 + u3;  
9 |         float uu = uu - (u0 + u1 + u2 + u3);  
9 |         float uu = uu / 4.0f;  
9 |         s0 = s0 + uu;  
9 |       }
```
Switching wavefronts

Now that I’ve been stepping only my wavefront, are the others still at the beginning of the kernel? Use info threads. This enables us to see the location of both host threads and GPU wavefronts...
Switching wavefronts

.. but where did all the other threads go? **Step allows other wavefronts to advance**, hence the rest of the waves in our kernel have already completed!
Rerunning rocgdb with scheduler-locking

Use ‘set scheduler-locking on’ will not continue the other waves.
Rerunning rocgdb with scheduler-locking

Now we see the rest of the GPU threads.
Can also type “thread <tid>” to examine one particular thread
Other tricks: export AMD_LOG_LEVEL=3

By setting the above environment variable, we can get a print of all API calls and more happening:
Other tricks: export AMD_LOG_LEVEL=3

By setting the above environment variable, we can get a print of all API calls and more happening:
Other tricks: export AMD_LOG_LEVEL=3

By setting the above environment variable, we can get a print of all API calls and more happening:
Other tricks: switching between host threads

Type ‘i th’ to see a list of all active host threads. Currently viewing thread 1 (default).
Other tricks: switching between host threads

Switching between the threads gives different stack traces
Other tricks: switching between host threads

Switching between the threads gives different stack traces
Other tricks: switching between host threads

Switching between the threads gives different stack traces
What to do if your program returns an error?

In a perfect world, all code should run successfully like this ...
What to do if your program returns an error?

... but every so often we come across GPU errors like this
What to do if your program returns an error?

Launch rocgdb and invoke the options shown in the picture:
What to do if your program returns an error?

We now see the offending line which invokes the error. Uncommenting out lines 20 and 31 will fix this particular issue.
And more...

ROCgdb has several other features and capabilities not covered in this presentation. See the following for much more:

- `/opt/rocm-4.2.0/share/doc/rocgdb/rocannotate.pdf`
- `/opt/rocm-4.2.0/share/doc/rocgdb/rocgdb.pdf`
- `/opt/rocm-4.2.0/share/doc/rocgdb/rocrefcard.pdf`
- `/opt/rocm-4.2.0/share/doc/rocgdb/rocstabs.pdf`
Part 2: Math Libraries
## Decoder ring: Math library equivalents

<table>
<thead>
<tr>
<th>CUBLAS</th>
<th>ROCBLAS</th>
<th>Basic Linear Algebra Subroutines</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUFFT</td>
<td>ROCFFT</td>
<td>Fast Fourier Transforms</td>
</tr>
<tr>
<td>THRUST</td>
<td>ROCTHRUST</td>
<td>Deep Learning Library</td>
</tr>
<tr>
<td>CUB</td>
<td>ROCPRIM</td>
<td>Optimized Parallel Primitives</td>
</tr>
<tr>
<td>EIGEN</td>
<td>EIGEN</td>
<td>C++ Template Library for Linear Algebra</td>
</tr>
</tbody>
</table>

More info at: [github.com/rocminfo/hip](https://github.com/rocminfo/hip) → HIP_PORTING_GUIDE.MD
AMD GPU Math Libraries

- A note on naming conventions:
  - roc* -> AMGCN library usually written in HIP
  - cu* -> NVIDIA PTX libraries
  - hip* -> usually interface layer on top of roc*/cu* backends

- hip* libraries:
  - Can be compiled by hipcc and can generate a call for the device you have:
    - hipcc->clang->AMD GCN ISA
    - hipcc->nvcc (inlined)->NVPTX
  - Just a thin wrapper that marshals calls off to a “backend” library:
    - corresponding roc* library backend containing optimized GCN
    - corresponding cu* library backend containing NVPTX for NVIDIA devices
  - E.g., hipBLAS is a marshalling library: 

  ![Diagram showing rocBLAS, hipBLAS, and cuBLAS]

  rocBLAS  
  hipBLAS  
  cuBLAS
AMD GPU Libraries: BLAS

- **rocBLAS** – `sudo apt install rocblas`
  - Source code: [https://github.com/ROCmSoftwarePlatform/rocBLAS](https://github.com/ROCmSoftwarePlatform/rocBLAS)
  - Basic linear algebra functionality
    - axpy, gemv, trsm, etc
  - Use hipBLAS if you need portability between AMD and NVIDIA devices

- **hipBLAS** - `sudo apt install hipblas`
  - Documentation: [https://github.com/ROCmSoftwarePlatform/hipBLAS/wiki/Exported-functions](https://github.com/ROCmSoftwarePlatform/hipBLAS/wiki/Exported-functions)
  - Use this if you need portability between AMD and NVIDIA
  - It is just a thin wrapper:
    - It can dispatch calls to rocBLAS for AMD devices
    - It can dispatch calls to cuBLAS for NVIDIA devices
AMD GPU Libraries: rocBLAS example

- **rocBLAS**
  - Level 1, 2, and 3 functionality
    - axpy, gemv, trsm, etc
  - Note: rocBLAS syntax matches BLAS closer than hipBLAS or cuBLAS
    - Use hipBLAS only if you need portability between AMD and NVIDIA devices
  - Link with: -lrocblas

```c
#include <rocblas.h>

int main(int argc, char **argv) {
    rocblas_int N = 500000;
    // Allocate device memory
double * dx, * dy;
    hipMalloc(&dx, sizeof(double) * N);
    hipMalloc(&dy, sizeof(double) * N);
    // Allocate host memory (and fill up the arrays) here
    std::vector<double> hx(N), hy(N);
    // Copy host arrays to device
    hipMemcpy(dx, hx.data(), sizeof(double) * N, hipMemcpyHostToDevice);
    hipMemcpy(dy, hy.data(), sizeof(double) * N, hipMemcpyHostToDevice);
    const double alpha = 1.0;
    rocblas_handle handle;
    rocblas_create_handle(&handle);
    rocblas_status status;
    status = rocblas_daxpy(handle, N, &alpha, dx, 1, dy, 1);
    rocblas_destroy_handle(handle);
    // Copy result back to host
    hipMemcpy(hy.data(), dy, sizeof(double) * N, hipMemcpyDeviceToHost);
    hipFree(dx);
    hipFree(dy);
    return 0;
}
```
AMD GPU Libraries: FFT

- **rocFFT** – `sudo apt install rocfft`
  - Source code: [https://github.com/ROCmSoftwarePlatform/rocFFT](https://github.com/ROCmSoftwarePlatform/rocFFT)
  - Implementation of Discrete Fourier Transforms (DFT), leveraging mathematical symmetries to reduce algorithmic complexity from $O(N^2)$ to $O(N \log N)$
  - Use **hipFFT** (`sudo apt install hipfft`) if you need portability between AMD and NVIDIA devices

- **Basic steps in rocFFT:**
  1. Initialize the library by calling `rocfft_setup()`
  2. Create a plan for the FFT needed
  3. Optionally allocate a work buffer
  4. Execute the plan
  5. Free the work buffer if needed
  6. Destroy the plan
  7. Terminate the library by calling `rocfft_cleanup()`
# AMD GPU Libraries: rocFFT example

```cpp
#include <iostream>
#include <vector>
#include "hip/hip_runtime_api.h"
#include "hip/hip_vector_types.h"
#include "rocfft.h"

int main()
{
    rocfft_setup();

    // Create HIP device buffer
    size_t N = 16;
    size_t Nbytes = N * sizeof(float2);
    float2 *x;
    hipMalloc(&x, Nbytes);

    // Initialize data
    std::vector<float2> cx(N);
    for (size_t i = 0; i < N; i++) {
        cx[i].x = 1; cx[i].y = -1;
    }

    // Copy data to device
    hipMemcpy(x, cx.data(), Nbytes, hipMemcpyHostToDevice);

    // Create rocFFT plan
    rocfft_plan plan = nullptr;
    size_t length = N;
    rocfft_plan_create(&plan, rocfft_placement_inplace,
        rocfft_transform_type_complex_forward, rocfft_precision_single,
        1, &length, 1, nullptr);

    // Check if the plan requires a work buffer
    size_t work_buf_size = 0;
    rocfft_plan_get_work_buffer_size(plan, &work_buf_size);
    void* work_buf = nullptr;
    rocfft_execution_info info = nullptr;
    if(work_buf_size) {
        rocfft_execution_info_create(&info);
        hipMalloc(&work_buf, work_buf_size);
        rocfft_execution_info_set_work_buffer(info, work_buf, work_buf_size);
    }

    // Execute plan
    rocfft_execute(plan, (void**) &x, nullptr, info);
    hipDeviceSynchronize();

    // Clean up work buffer
    if(work_buf_size) {
        hipFree(work_buf);
        rocfft_execution_info_destroy(info);
    }

    // Destroy plan
    rocfft_plan_destroy(plan);

    // Copy result back to host
    std::vector<float2> y(N);
    hipMemcpy(y.data(), x, Nbytes, hipMemcpyDeviceToHost);

    // Free device buffer
    hipFree(x);
    rocfft_cleanup();
    return 0;
}
```
Some Links to Key Libraries

- **BLAS**
  - rocBLAS ([https://github.com/ROCmSoftwarePlatform/rocBLAS](https://github.com/ROCmSoftwarePlatform/rocBLAS))
  - hipBLAS ([https://github.com/ROCmSoftwarePlatform/hipBLAS](https://github.com/ROCmSoftwarePlatform/hipBLAS))

- **FFT**s
  - rocFFT ([https://github.com/ROCmSoftwarePlatform/rocFFT](https://github.com/ROCmSoftwarePlatform/rocFFT))

- **Random number generation**
  - rocRAND ([https://github.com/ROCmSoftwarePlatform/rocRAND](https://github.com/ROCmSoftwarePlatform/rocRAND))
  - hipRAND ([https://github.com/ROCmSoftwarePlatform/hipRAND](https://github.com/ROCmSoftwarePlatform/hipRAND))

- **Sparse linear algebra**
  - rocSPARSE ([https://github.com/ROCmSoftwarePlatform/rocSPARSE](https://github.com/ROCmSoftwarePlatform/rocSPARSE))
  - hipSPARSE ([https://github.com/ROCmSoftwarePlatform/hipSPARSE](https://github.com/ROCmSoftwarePlatform/hipSPARSE))

- **Iterative solvers**
  - rocALUTION ([https://github.com/ROCmSoftwarePlatform/rocALUTION](https://github.com/ROCmSoftwarePlatform/rocALUTION))

- **Parallel primitives**
  - rocPRIM ([https://github.com/ROCmSoftwarePlatform/rocPRIM](https://github.com/ROCmSoftwarePlatform/rocPRIM))
  - hipCUB ([https://github.com/ROCmSoftwarePlatform/hipCUB](https://github.com/ROCmSoftwarePlatform/hipCUB))
More links to key libraries

Machine Learning libraries and Frameworks

- Tensorflow: https://github.com/ROCmSoftwarePlatform/tensorflow-upstream
- Pytorch: https://github.com/ROCmSoftwarePlatform/pytorch
- MIOpen (similar to cuDNN): https://github.com/ROCmSoftwarePlatform/MIOpen
- Tensile: https://github.com/ROCmSoftwarePlatform/Tensile
- RCCL (ROCm analogue of NCCL): https://github.com/ROCmSoftwarePlatform/rccl
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 for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, 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.