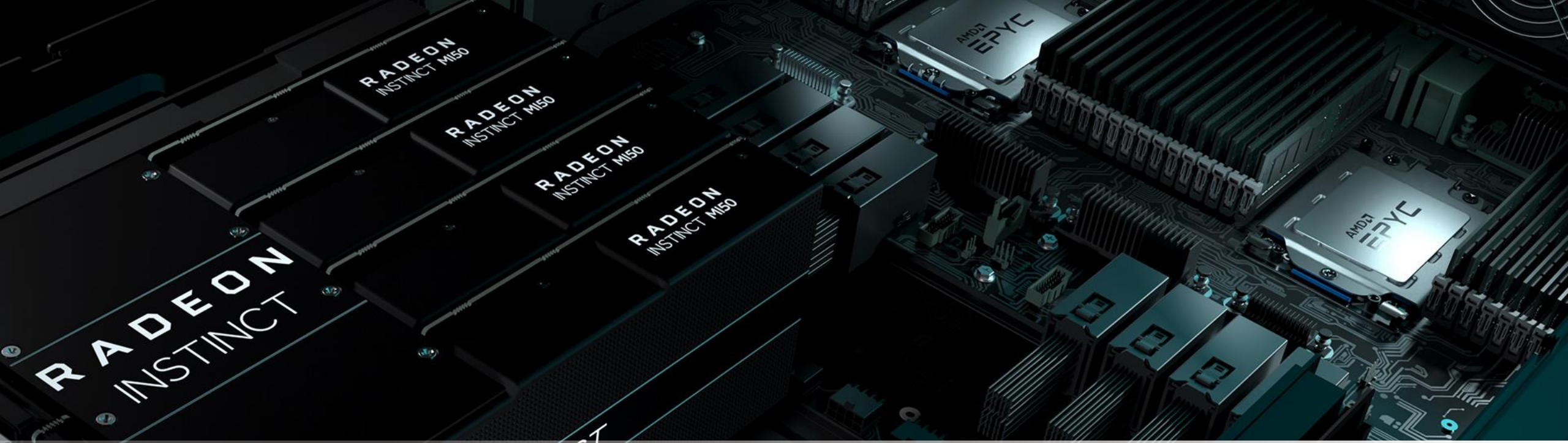


ATPESC 2021 – Tools Track

ROC-profiler and debugger: An Overview of ROCm Tools

Suyash Tandon, Justin Chang, Julio Maia, Noel Chalmers, Paul T. Bauman, Nicholas Curtis, Nicholas Malaya, Damon McDougall, Rene van Oostrum, Noah Wolfe

AMD

AMD EPYC | AMD RADEON INSTINCT | ROCm

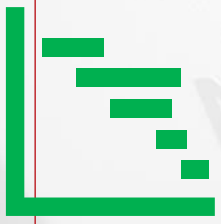
Agenda

1. ROC-profiler
2. ROC-debugger

*More information can be fund in the official ROCm documentation

Profiling

AMD GPU Profiling



ROC-profiler (or simply rocprof) is the command line front-end for AMD GPU profiling library

- Repo: https://github.com/ROCm-Developer-Tools/rocprofiler

rocprof contains the central components allowing the collection of application tracing and counter collection

- Under constant development

Provided in the ROCm releases

The output of rocprof can be visualized using the chrome browser with chrome tracing

Getting started with rocprof

To get help:

- `$ /opt/rocm/bin/rocprof -h`

Useful housekeeping flags:

- `--timestamp <on|off>` : turn on/off gpu kernel timestamps
- `--basenames <on|off>`: turn on/off truncating gpu kernel names (i.e., removing template parameters and argument types)
- `-o <output csv file>`: Direct counter information to a particular file name
- `-d <data directory>`: Send profiling data to a particular directory
- `-t <temporary directory>`: Change the directory where data files typically created in /tmp are placed. This allows you to save these temporary files

Flags directing rocprofiler activity:

- `-i <input.txt|.xml>`: specify an input file (note the output files will be named input.*)
- `--hsa-trace`: to trace GPU Kernels, host HSA events (more later) and HIP memory copies.
- `--hip-trace`: to trace HIP API calls
- `--roctx-trace`: to trace roctx markers

Advanced usage

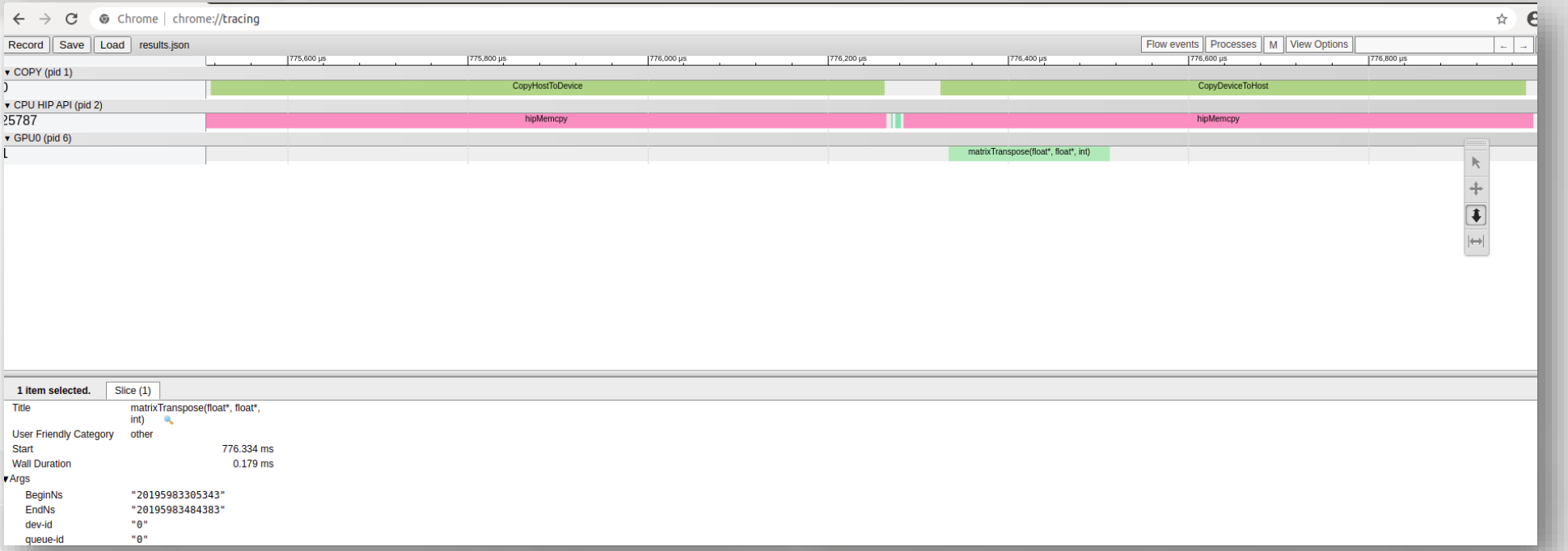
- `-m <metric file>`: Allows the user to define and collect custom metrics. See rocprofiler/test/tool/*.xml on GitHub for examples

Collecting traces with rocprof

rocprof can collect a variety of trace event types and generate timelines in JSON format for use with chrome-tracing, currently:

Trace Event	rocprof Trace Mode
HIP API call	--hip-trace
GPU Kernels	--hip-trace
Host <-> Device Memory copies	--hip-trace
CPU HSA Calls	--hsa-trace
User code markers	--roctx-trace

Collecting hip-trace with rocprof

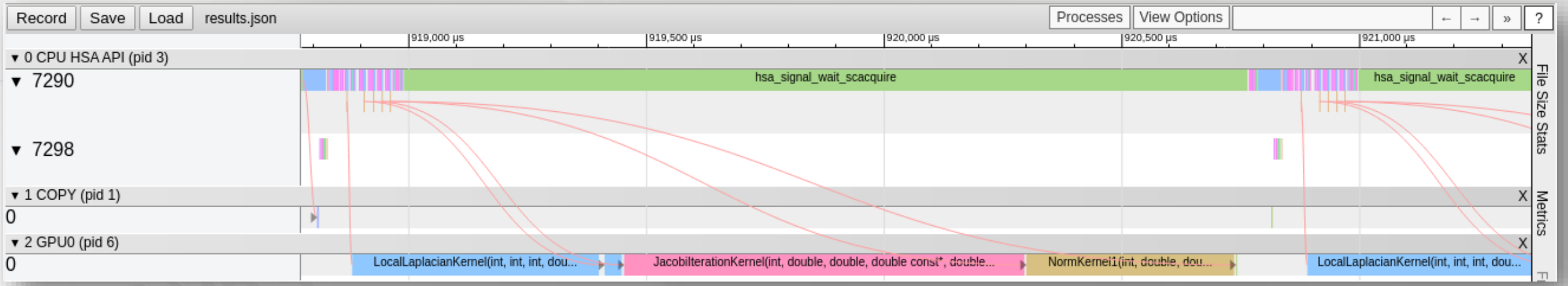



```
$ rocprof --hip-trace <app with arguments>
```

This will output a .json file that can be visualized using the chrome browser

Go to chrome://tracing and then load the .json file

- The trace will display HIP calls, mem copies, kernels



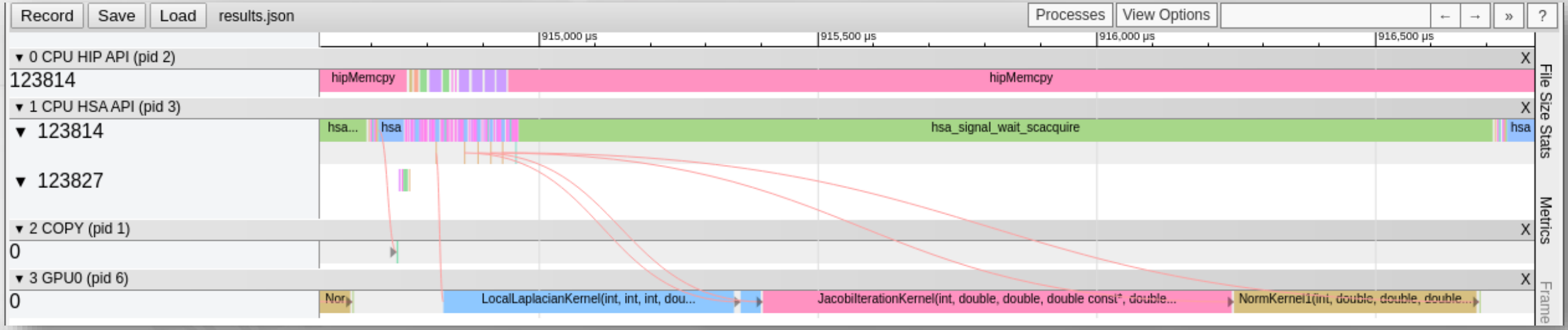
Collecting hsa-trace with rocprof

```
$ rocprof --hsa-trace <app with arguments>
```

This will output a .json file that can be visualized using the chrome browser

Go to chrome://tracing and then load the .json file

- The trace will display copies, hsa signals, and kernel calls
- Slowest trace mode – Use with cautions



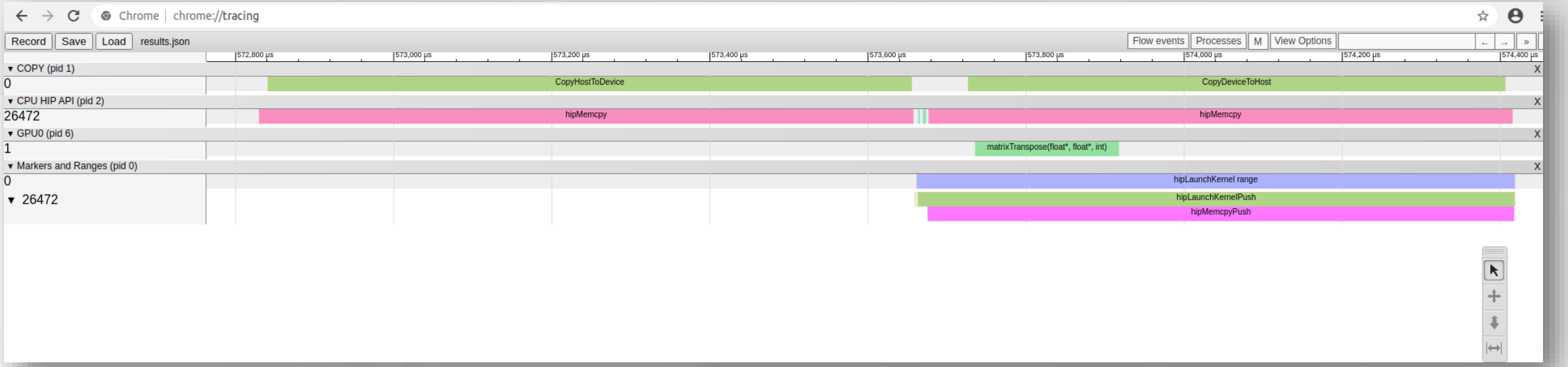
Collecting multiple trace modes with rocprof

```
$ rocprof --hsa-trace --hip-trace <app with arguments>
```

This will output a .json file that can be visualized using the chrome browser

Go to chrome://tracing and then load the .json file

- The trace will display HIP calls, copies, hsa signals, and kernel calls



Collecting user-code marker using rocTX with rocprof

```
$ rocprof --hip-trace --roctx-trace <app with arguments>
```

See MatrixTranspose.cpp example on roctracer GitHub page for sample in-code usage

This will output a .json file that can be visualized using the chrome browser

Collecting hardware counters with rocprof

rocprof can collect a number of hardware counters and derived counters

- `$/opt/rocm/bin/rocprof --list-basic`
- `$/opt/rocm/bin/rocprof --list-derived`

Specify counters in a counter file. For example:

- `$/opt/rocm/bin/rocprof -i my_counters.txt <app with args>`
- `$cat my_counters.txt`

  pmc : Wavefronts VALUInsts VFetchInsts VWriteInsts VALUUtilization VALUBusy WriteSize

  pmc : SALUInsts SFetchInsts LDSInsts FlatLDSInsts GDSInsts SALUBusy FetchSize

  pmc : L2CacheHit MemUnitBusy MemUnitStalled WriteUnitStalled ALUStalledByLDS LDSBankConflict...
- A limited number of counters can be collected during a specific pass of code
  - Each line in the counter file will be collected in one pass
  - You will receive an error suggesting alternative counter ordering if you have too many / conflicting counters on one line
- A `.csv` file will be created by this command containing all the requested counters

Commonly used counters

VALUUtilization	• The percentage of ALUs active in a wave. Low VALUUtilization is likely due to high divergence or a poorly sized grid
VALUBusy	• The percentage of GPUTime the vector ALU instructions are processed. Can be thought of as something like compute utilization
FetchSize	• The total kilobytes fetched from global memory
WriteSize	• The total kilobytes written to global memory
L2CacheHit	• The percentage of fetch, write, atomic, and other instructions that hit the data in L2 cache
MemUnitBusy	• The percentage of GPUTime the memory unit is active. The result includes the stall time
MemUnitStalled	• The percentage of GPUTime the memory unit is stalled
WriteUnitStalled	• The percentage of GPUTime the write unit is stalled
Full list at	• https://github.com/ROCm-Developer-Tools/rocprofiler/blob/amd-master/test/tool/metrics.xml

Performance counters tips and tricks

GPU Hardware counters are global

- Kernel dispatches are serialized to ensure that only one dispatch is ever in flight
- It is recommended that no other applications are running that use the GPU when collecting performance counters

Use "`--basenames on`" which will report only kernel names, leaving off kernel arguments.

How do you time a kernel's duration?

- `$rocprof --timestamps on -i my_counters.txt <app with args>`
- This produces four times: DispatchNs, BeginNs, EndNs, and CompleteNs
- Closest thing to a kernel duration: EndNs - BeginNs
- If you run with `--stats` the resultant results file will automatically include a column that calculates kernel duration
  - Note: the duration is aggregated over repeated calls to the same kernel

Collecting counters and traces

on multiple MPI ranks

- rocprof can collect counters and traces for multiple MPI ranks.
- Say you want to profile an application usually called like this:

```
mpiexec –np <n> <app with args>
```

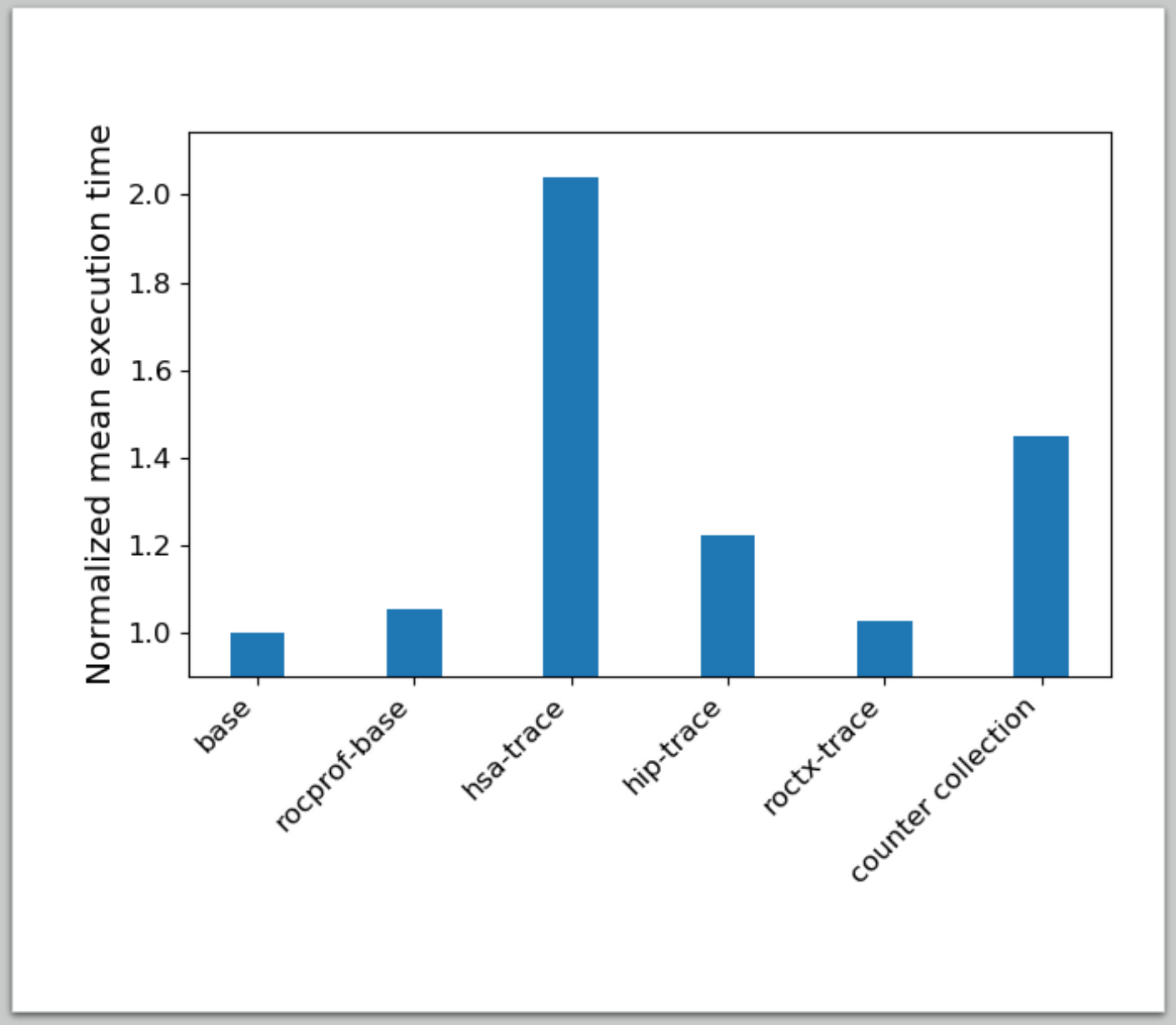
- Then invoke the profiler by executing:

```
rocprof --hip-trace mpiexec -np <n> <app with args>
```

- This will produce a single unified CSV file for all ranks
- Multi-node profiling currently isn’t supported

Profiling overhead

Simple estimation of profiling overhead, obtained via wall-clock timing of entire application run via Linux ‘time’ utility:



Debugger

Intro to ROCgdb

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.

Preparing the code for the debugger

Use any optimization level you like,

For example: `-O3`

Have ROCm load code objects at initialization:

```
export HIP_ENABLE_DEFERRED_LOADING=0
```

Add the flags:

`-ggdb`

Optionally print even more useful information on API calls

```
export AMD_LOG_LEVEL=3
```

Example of what the compile options may look like…

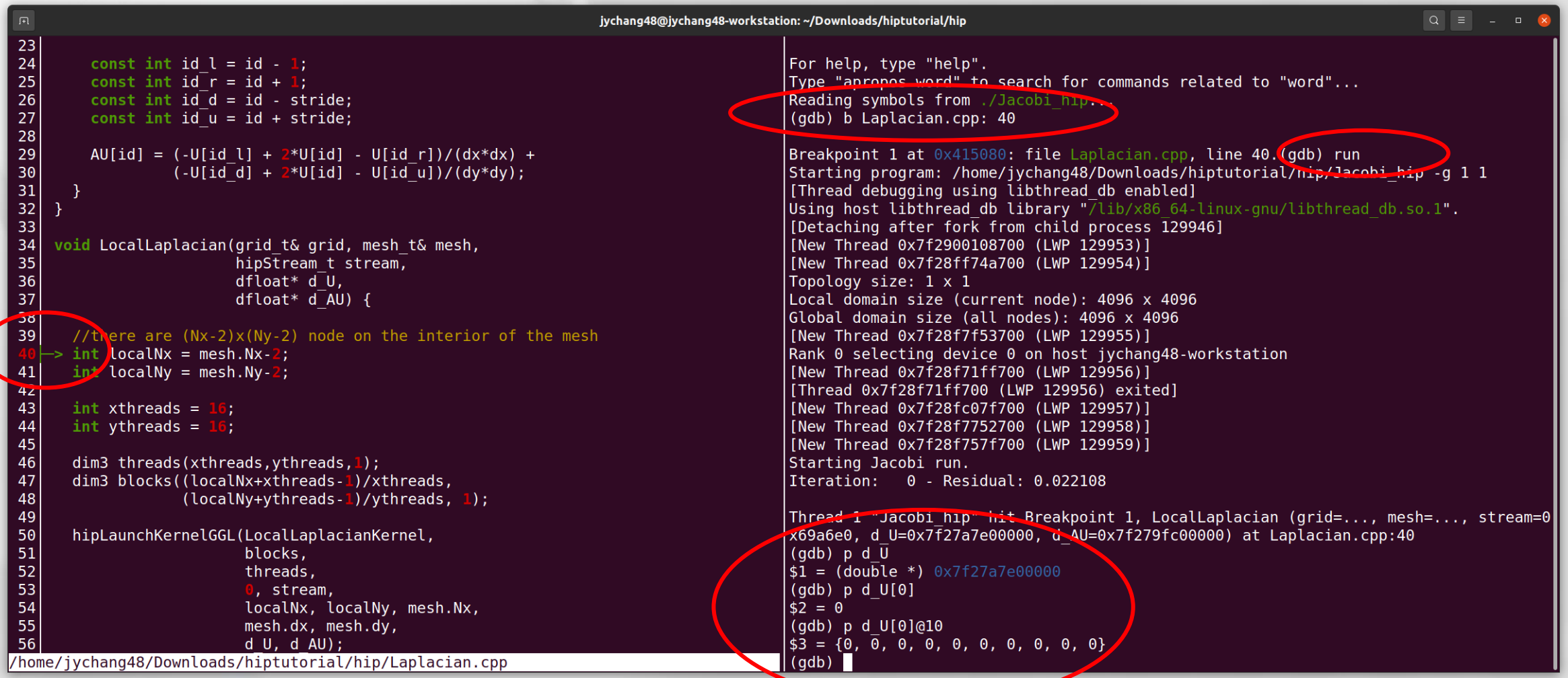
```
mpic++ -I/usr/lib/x86_64-linux-gnu/openmpi/include/openmpi –L/usr/lib/x86_64-linux-
gnu/openmpi/include -pthread –O3 –g –ggdb –fPIC –std=c++11 \ –march=native –Wall –
I/opt/rocm/roctracer/include  –I"/opt/rocm-4.2.0/hip/include" -
I"/opt/rocm/llvm/bin/../lib/clang/12.0.0" –I/opt/rocm/hsa/include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


What can it do?

- 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

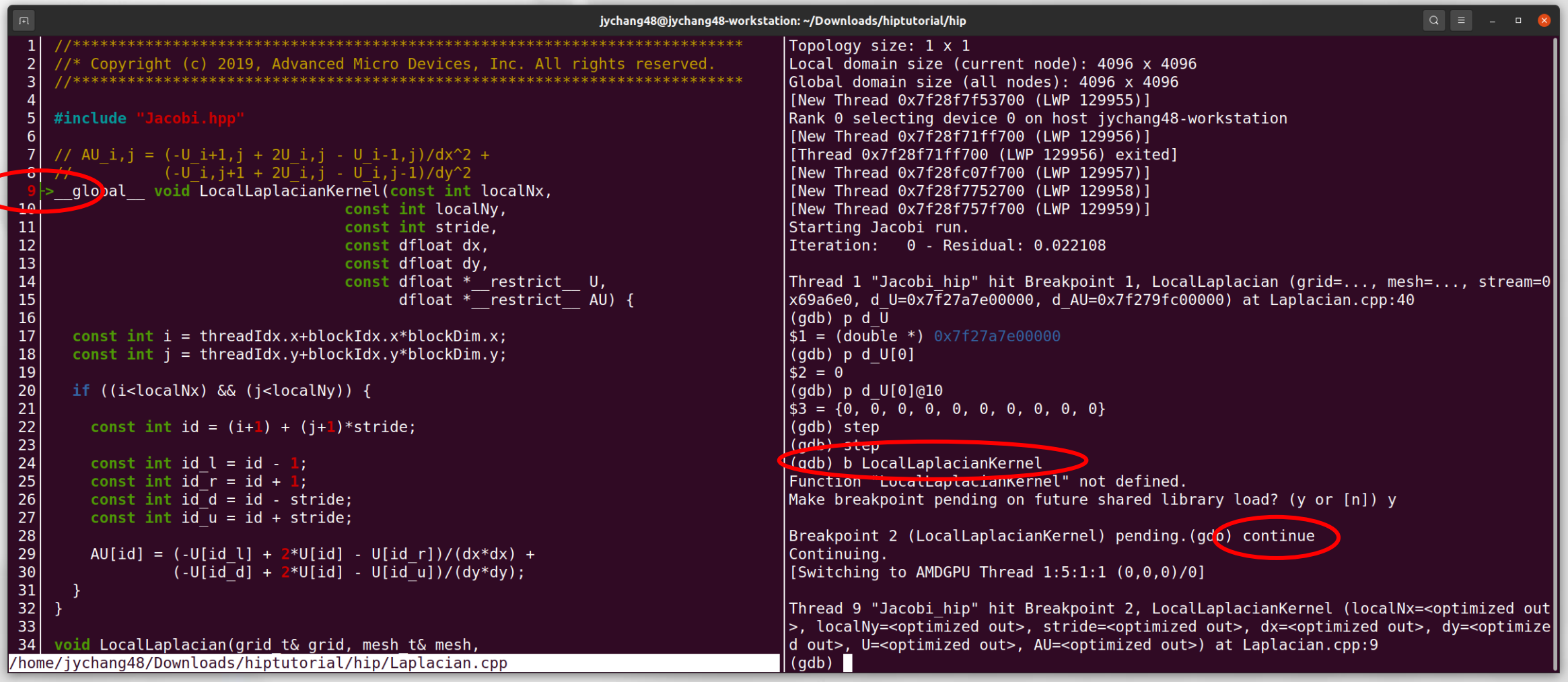
Setting a breakpoint in host code

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



Setting a breakpoint in device kernel

Invoke ‘b’ or ‘break’ to the device kernel of interest:



Examine switching context to new thread

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

jychang48@jychang48-workstation: ~/Downloads/hiptutorial/hip

```
 1  //********************************************************************************
 2  //* Copyright (c) 2019, Advanced Micro Devices, Inc. All rights reserved.
 3  //********************************************************************************
 4
 5  #include "Jacobi.hpp"
 6
 7  // AU_i,j = (-U_i+1,j + 2U_i,j - U_i-1,j)/dx^2 +
 8  //          (-U_i,j+1 + 2U_i,j - U_i,j-1)/dy^2
 9  __global__ void LocalLaplacianKernel(const int localNx,
10                                       const int localNy,
11                                       const int stride,
12                                       const dfloat dx,
13                                       const dfloat dy,
14                                       const dfloat *__restrict__ U,
15                                             dfloat *__restrict__ AU) {
16
17    const int i = threadIdx.x+blockIdx.x*blockDim.x;
18    const int j = threadIdx.y+blockIdx.y*blockDim.y;
19
20    if ((i<localNx) && (j<localNy)) {
21
22      const int id = (i+1) + (j+1)*stride;
23
24      const int id_l = id - 1;
25      const int id_r = id + 1;
26      const int id_d = id - stride;
27      const int id_u = id + stride;
28
29      AU[id] = (-U[id_l] + 2*U[id] - U[id_r])/(dx*dx) +
30               (-U[id_d] + 2*U[id] - U[id_u])/(dy*dy);
31    }
32  }
33
34  void LocalLaplacian(grid_t& grid, mesh_t& mesh,
/home/jychang48/Downloads/hiptutorial/hip/Laplacian.cpp
```

```
$2 = 0
(gdb) p d_U[0]@10
$3 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
(gdb) step
(gdb) step
(gdb) b LocalLaplacianKernel
Function "LocalLaplacianKernel" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y

Breakpoint 2 (LocalLaplacianKernel) pending.(gdb) continue
Continuing.
[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=<optimize
d out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9
(gdb) step
[Switching to AMDGPU Thread 1:5:1:1287 (65,1,0)/2]

Thread 1295 "Jacobi_hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized
out>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optim
ized out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9
(gdb) step
[Switching to AMDGPU Thread 1:5:1:1269 (61,1,0)/0]

Thread 1277 "Jacobi_hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized
out>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optim
ized out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9
(gdb) step
[Switching to AMDGPU Thread 1:5:1:363 (90,0,0)/2]

Thread 371 "Jacobi_hip" hit Breakpoint 2, LocalLaplacianKernel (localNx=<optimized o
ut>, localNy=<optimized out>, stride=<optimized out>, dx=<optimized out>, dy=<optimi
zed out>, U=<optimized out>, AU=<optimized out>) at Laplacian.cpp:9
(gdb)
```

Examine switching context to new thread

AMDGPU Thread agent-id:queue-id:dispatch-num:wave-id **(work-group-z,work-group-y,work-group-x)/work-group-thread-index**

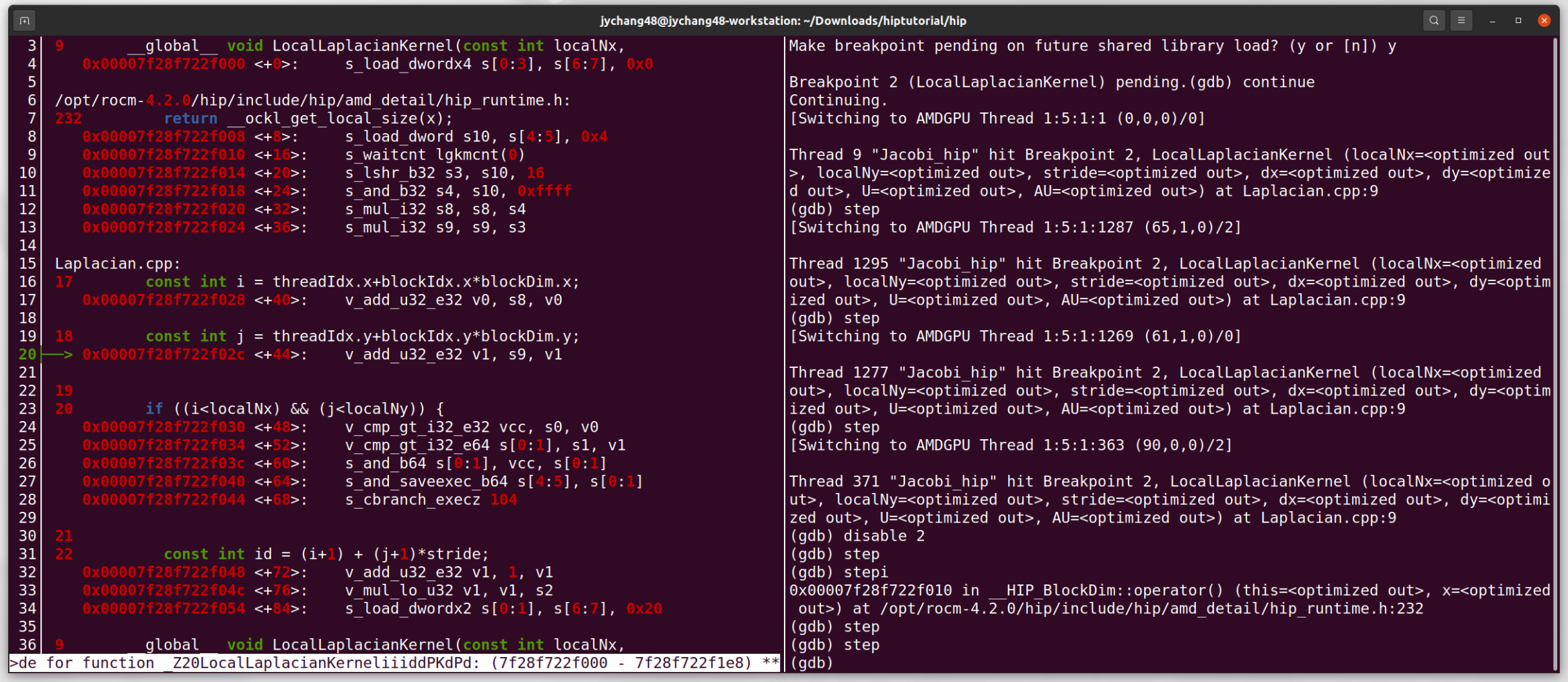
```
[Switching to AMDGPU Thread 1:5:1:1 (0,0,0)/0]

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

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

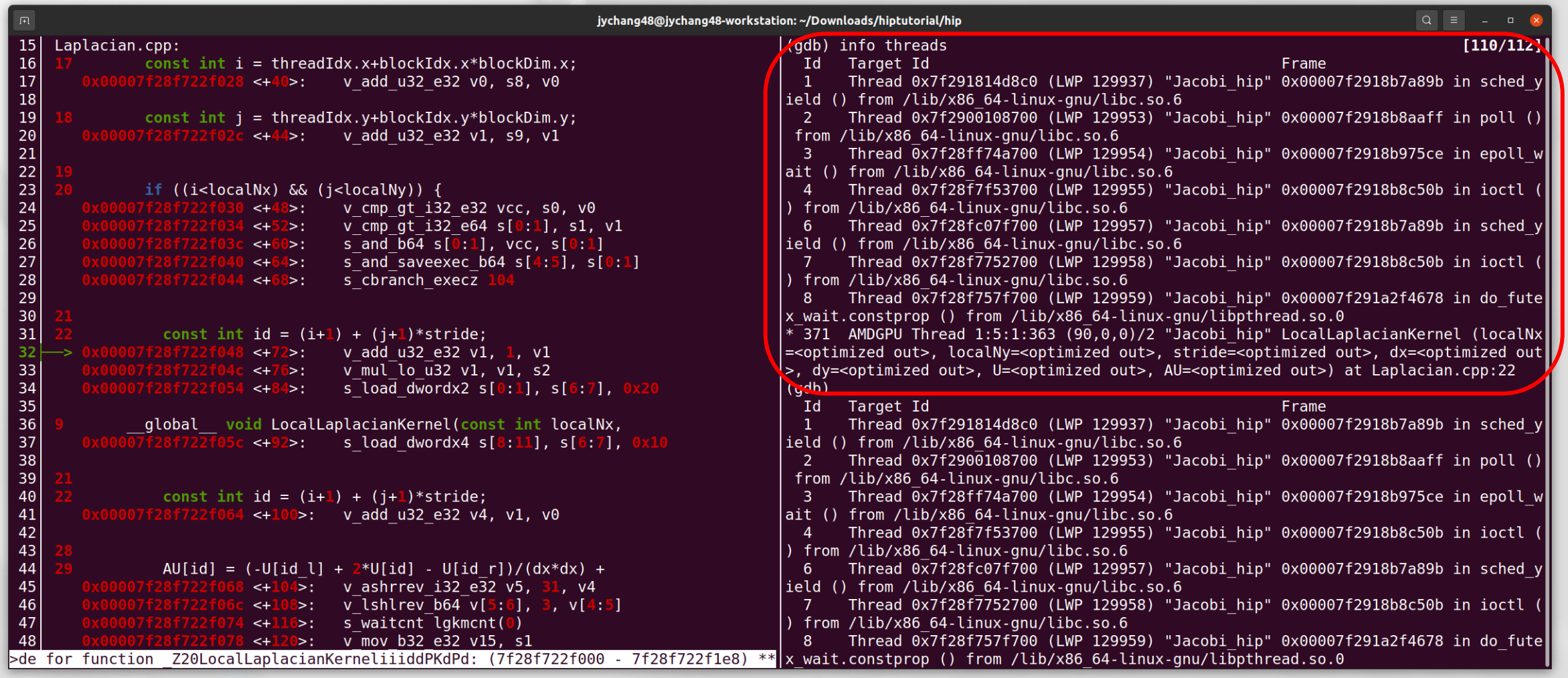
Examine the ISA with rocgdb using cgdb

Using cgdb, type
ESC -> :set dis -> ENTER



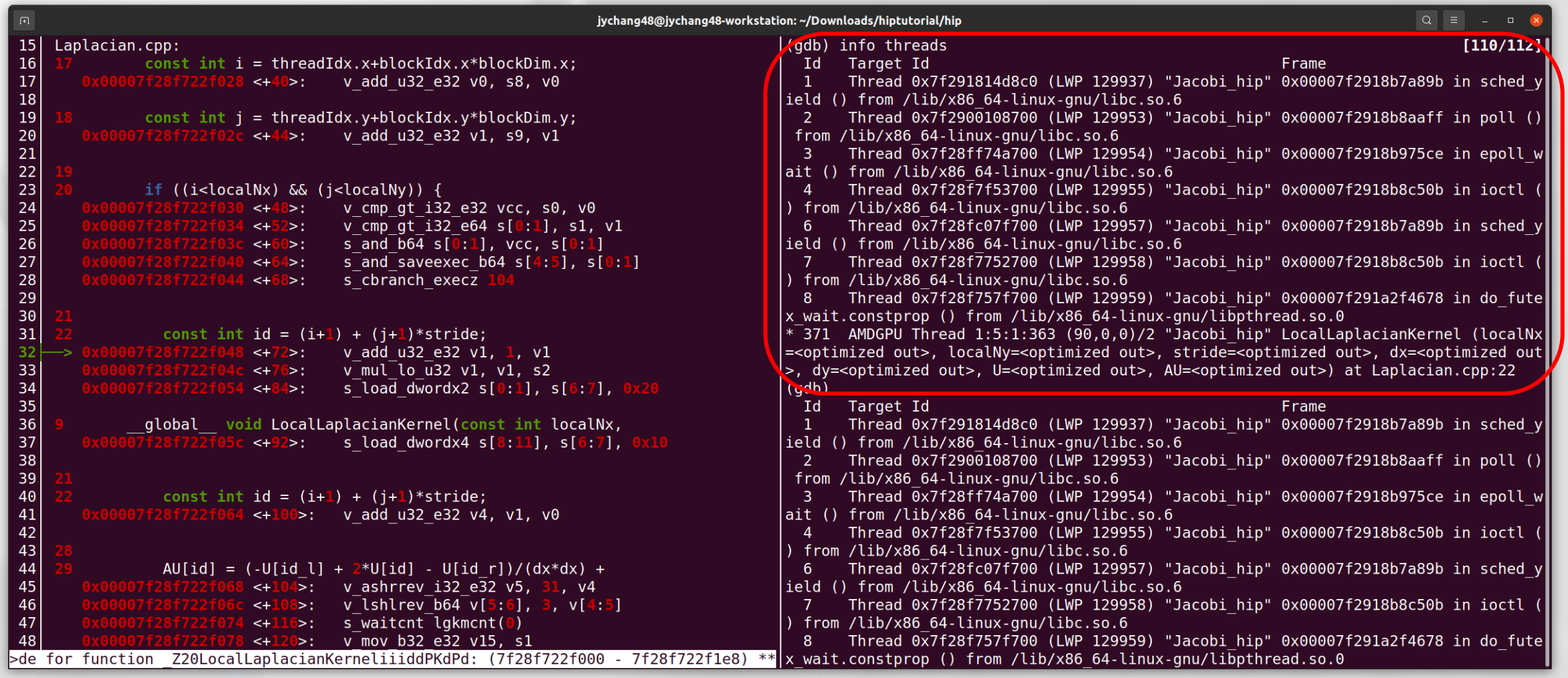
Switching wavefronts

Use info threads to see the location of both host threads and GPU wavefronts



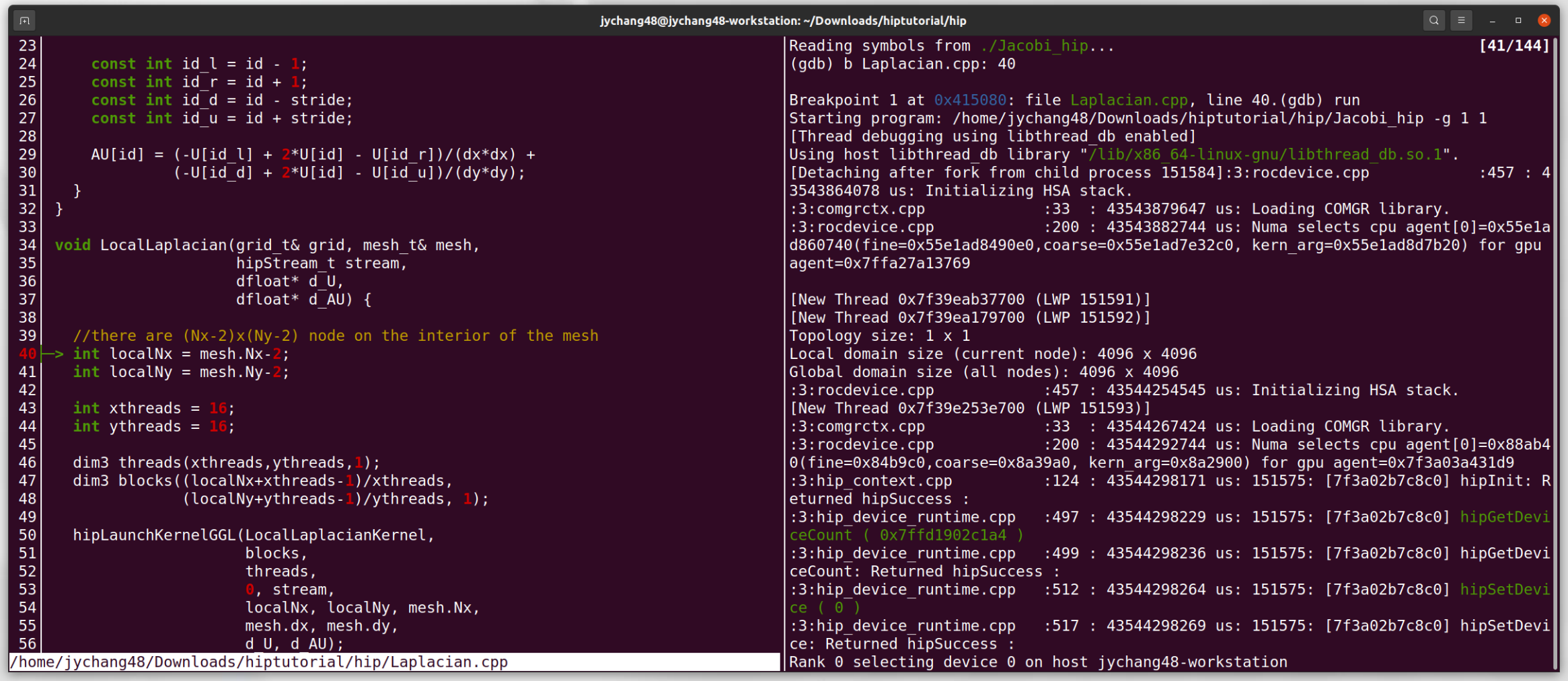
Switching wavefronts

Or use `thread <tid>` to examine one particular thread



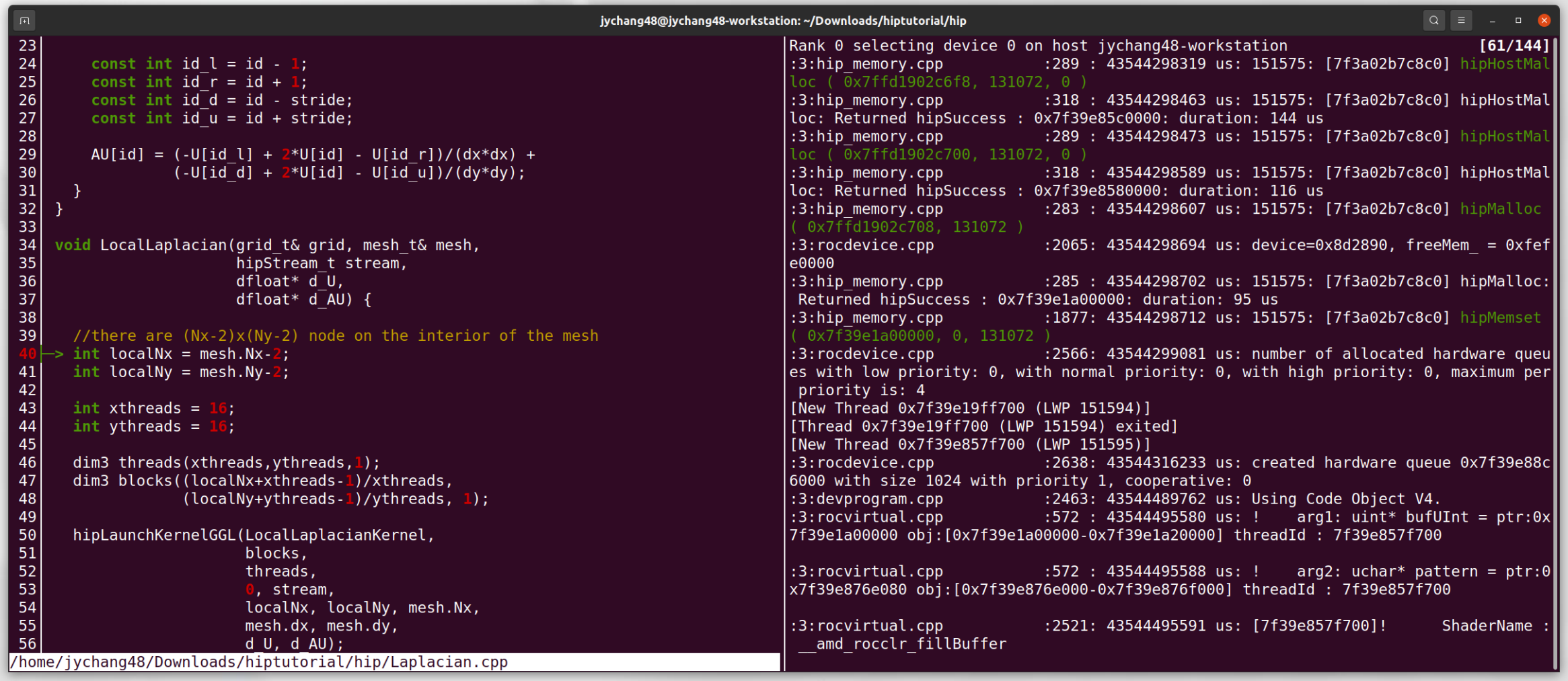
Other tips and tricks with rocgdb

Use `export AMD_LOG_LEVEL=3` to print all API calls and more happening



Other tips and tricks with rocgdb

Use `export AMD_LOG_LEVEL=3` to print all API calls and more happening



Other tips and tricks with rocgdb

Use `i th` to see a list of all active host threads. Currently viewing thread 1 (default)

jychang48@jychang48-workstation: ~/Downloads/hiptutorial/hip

```
23
24      const int id_l = id - 1;
25      const int id_r = id + 1;
26      const int id_d = id - stride;
27      const int id_u = id + stride;
28
29      AU[id] = (-U[id_l] + 2*U[id] - U[id_r])/(dx*dx) +
30               (-U[id_d] + 2*U[id] - U[id_u])/(dy*dy);
31    }
32  }
33
34  void LocalLaplacian(grid_t& grid, mesh_t& mesh,
35                      hipStream_t stream,
36                      dfloat* d_U,
37                      dfloat* d_AU) {
38
39    //there are (Nx-2)x(Ny-2) node on the interior of the mesh
40--> int localNx = mesh.Nx-2;
41    int localNy = mesh.Ny-2;
42
43    int xthreads = 16;
44    int ythreads = 16;
45
46    dim3 threads(xthreads,ythreads,1);
47    dim3 blocks((localNx+xthreads-1)/xthreads,
48                (localNy+ythreads-1)/ythreads, 1);
49
50    hipLaunchKernelGGL(LocalLaplacianKernel,
51                       blocks,
52                       threads,
53                       0, stream,
54                       localNx, localNy, mesh.Nx,
55                       mesh.dx, mesh.dy,
56                       d_U, d_AU);
/home/jychang48/Downloads/hiptutorial/hip/Laplacian.cpp
```

```
:3:rocvirtual.cpp           :2521: 43544718546 us: [7f39e8446700]!      ShaderName :
 _Z11NormKernel2iPKdPd

:3:hip_memory.cpp           :331 : 43544719486 us: 151575: [7f3a02b7c8c0] hipMemcpy:
 Returned hipSuccess : : duration: 957 us
Iteration:    0 - Residual: 0.022108
:3:hip_device_runtime.cpp   :458 : 43544719507 us: 151575: [7f3a02b7c8c0] hipDeviceS
ynchronize (  )
:3:hip_device_runtime.cpp   :470 : 43544719512 us: 151575: [7f3a02b7c8c0] hipDeviceS
ynchronize: Returned hipSuccess :
:3:hip_event.cpp            :310 : 43544719523 us: 151575: [7f3a02b7c8c0] hipEventRe
cord ( event:0x948d00, stream:0x789710 )
:3:hip_event.cpp            :352 : 43544719530 us: 151575: [7f3a02b7c8c0] hipEventRe
cord: Returned hipSuccess :

Thread 1 "Jacobi_hip" hit Breakpoint 1, LocalLaplacian (grid=..., mesh=..., stream=0
x789710, d_U=0x7f3893e00000, d_AU=0x7f388bc00000) at Laplacian.cpp:40
(gdb) i th
  Id   Target Id                                          Frame
* 1    Thread 0x7f3a02b7c8c0 (LWP 151575) "Jacobi_hip" LocalLaplacian (grid=..., mes
h=..., stream=0x789710, d_U=0x7f3893e00000, d_AU=0x7f388bc00000) at Laplacian.cpp:40
  2    Thread 0x7f39eab37700 (LWP 151591) "Jacobi_hip" 0x00007f3a035b9aff in poll ()
 from /lib/x86_64-linux-gnu/libc.so.6
  3    Thread 0x7f39ea179700 (LWP 151592) "Jacobi_hip" 0x00007f3a035c65ce in epoll_w
ait () from /lib/x86_64-linux-gnu/libc.so.6
  4    Thread 0x7f39e253e700 (LWP 151593) "Jacobi_hip" 0x00007f3a035bb50b in ioctl (
) from /lib/x86_64-linux-gnu/libc.so.6
  6    Thread 0x7f39e857f700 (LWP 151595) "Jacobi_hip" 0x00007f3a04d23678 in do_fute
x_wait.constprop () from /lib/x86_64-linux-gnu/libpthread.so.0
  7    Thread 0x7f39e8446700 (LWP 151596) "Jacobi_hip" 0x00007f3a04d23678 in do_fute
x_wait.constprop () from /lib/x86_64-linux-gnu/libpthread.so.0
  8    Thread 0x7f39e827f700 (LWP 151597) "Jacobi_hip" 0x00007f3a04d23678 in do_fute
x_wait.constprop () from /lib/x86_64-linux-gnu/libpthread.so.0
(gdb)
```


And more…

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

https://github.com/RadeonOpenCompute/ROCm/blob/master/ROCm_Debugger_User_Guide_v4.2.pdf
/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

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 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.

Third-party content is licensed to you directly by the third party that owns the content and is not licensed to you by AMD. ALL LINKED THIRD-PARTY CONTENT IS PROVIDED "AS IS" WITHOUT A WARRANTY OF ANY KIND. USE OF SUCH THIRD-PARTY CONTENT IS DONE AT YOUR SOLE DISCRETION AND UNDER NO CIRCUMSTANCES WILL AMD BE LIABLE TO YOU FOR ANY THIRD-PARTY CONTENT. YOU ASSUME ALL RISK AND ARE SOLELY RESPONSIBLE FOR ANY DAMAGES THAT MAY ARISE FROM YOUR USE OF THIRD-PARTY CONTENT.


AMD