Debugging GPU-Accelerated Applications with NVIDIA Developer Tools
Andrew Gontarek, Software Engineer | ALCF Developer Sessions 11/30/2022
Introduction to the NVIDIA debugging toolchest

Overview

- NVIDIA HPC SDK
  - A comprehensive suite of compilers, libraries and tools for HPC
  - Provided by nvhpc module
    - nvhpc/21.9 is default on Polaris

- Bundled with the HPC SDK is a debugging toolchest
  - CUDA-GDB
    - Interactive thread-based debugger
  - Compute Sanitizer
    - Functional correctness checking suite

<table>
<thead>
<tr>
<th>DEVELOPMENT</th>
<th>ANALYSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Models</td>
<td>Profilers</td>
</tr>
<tr>
<td>Standard C++ &amp; Fortran</td>
<td>Nvidia</td>
</tr>
<tr>
<td>OpenACC &amp; OpenMP</td>
<td>Systems</td>
</tr>
<tr>
<td>CUDA</td>
<td>Compute</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compilers</th>
<th>Debugger</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvcc</td>
<td>cuda-gdb</td>
</tr>
<tr>
<td>mpc</td>
<td></td>
</tr>
<tr>
<td>mpc++</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core Libraries</th>
<th>Communication Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>libc6++</td>
<td>HPC-X</td>
</tr>
<tr>
<td>Thrust</td>
<td>MPI</td>
</tr>
<tr>
<td>CUB</td>
<td>UCX</td>
</tr>
<tr>
<td>cuBLAS</td>
<td>SHMEM</td>
</tr>
<tr>
<td>cuSOLVER</td>
<td>SHARP</td>
</tr>
<tr>
<td>cuFFT</td>
<td>HCCL</td>
</tr>
<tr>
<td>cuRAND</td>
<td>NVSHMEM</td>
</tr>
<tr>
<td>NCCL</td>
<td></td>
</tr>
</tbody>
</table>

| Math Libraries | |
|----------------| |
| cuBLAS | |
| cuSOLVER | |
| cuRAND | |
Overview: CUDA-GDB
Overview: CUDA-GDB
What is it?

- Built on the familiar GDB debugger!
  - Ease-of-use: Users already familiar with gdb
  - GPU debugging provides a similar logical experience
  - Existing C/C++/Fortran support
  - Seamless experience between host (CPU) and device (GPU) debugging
  - Support for CUDA/OptiX/OpenACC/OpenMP/etc source level device code
  - Support for SASS disassembly
  - Various command extensions unique to CUDA-GDB

- Interactive CLI based tool
- Provides reactive debugging of CUDA kernels
  - CUDA Runtime errors
  - Debugging when exceptions occur
  - Logic errors producing incorrect answers

- Post-mortem debugging with corefiles
  - Coredump capture enabled via environment variables
Overview: CUDA-GDB
Quickstart

• On Polaris
  • Provided by PATH from module nvhpc/21.9 (default)

    agontarek@polaris-login-01:~> which cuda-gdb
    /opt/nvidia/hpc_sdk/Linux_x86_64/21.9/compilers/bin/cuda-gdb

• Latest documentation: https://docs.nvidia.com/cuda/cuda-gdb/index.html
• Tips and Tricks: https://docs.nvidia.com/cuda/cuda-gdb/index.html#advanced-settings
Overview: CUDA-GDB

Quickstart

- Recompile application for debugging
  - When compiling with `nvcc`:
    - Provide `-g` for host (CPU) debugging
    - Provide `-G` for device (GPU) debugging
    ```
    $ nvcc -g -G -o foo foo.cu
    ```
  - Using `-lineinfo` will allow debugging of optimized code
    - Lacks `.debug_info` sections
      - No symbolic debugging
    - Debugging optimized code can be a challenging experience
    - Check your compiler manual!
      - Command line arguments can vary by compiler
  - Pascal+ GPUs have an improved debugging experience
    - Out of scope for this presentation
    - Feature support listed in the CUDA-GDB manual

- HPC features of interest
  - Mutli-GPU debugging is supported on same node
  - Multiple CUDA-GDB instances can debug multiple processes running on same node

- HPC features of interest (cont.)
  - Limited CUDA-GDB support for CUDA Multi Process Server (MPS)
    - [https://docs.nvidia.com/deploy/mps/index.html#topic_3_3_6_1](https://docs.nvidia.com/deploy/mps/index.html#topic_3_3_6_1)
  - Use `CUDA_VISIBLE_DEVICES` env var to select which GPUs are available to the application
  - CUDA Lazy Loading feature can speed up debugging times significantly (10x or more)
    ```
    $ export CUDA_MODULE_LOADING=lazy
    ```
    - Requires CUDA Toolkit 11.8+ and CUDA driver r520+
    - Defers loading cubins until first use
    - Especially helpful for applications linked against large math libs
  - CUDA-GDB uses `TMPDIR` to write temporary files
    - Defaults to `/tmp` if `TMPDIR` unset
    - Directory required to be writable
    - Needs to be the same for both application and CUDA-GDB
Overview: CUDA-GDB

Terminology

• Assume: familiarity with the CUDA programming model
• Exposes both logical and physical concepts to user

• Logical
  • Kernel
    • A function executed in parallel on the device
    • Executed as a grid of blocks of threads
      • Specified by the <<<...>>> syntax
  • Block
    • Consists of threads – 1024 threads max
    • 3-dimensional coordinate
      • dim3 named blockIdx
      • Bounded by gridDim
  • Thread
    • Smallest unit of work
    • 3-dimensional coordinate
      • dim3 named threadIdx
      • Bounded by blockDim

• Physical
  • Device
    • CUDA capable GPUs
    • Comprised of many SMs
  • SM
    • Streaming multiprocessor
    • Executes block(s) in warp sized chunks
  • Warp
    • Group of 32 lanes
  • Lane
    • Core that executes CUDA thread
Overview: CUDA-GDB
Terminology

• CUDA focus
  • Most CUDA-GDB commands apply to a single thread in focus
    • Focus can be host or device thread
  • Breakpoints or exceptions inside a CUDA kernel will automatically switch to device focus

• CUDA focus (cont.)
  • Thread identifier (logical)
  • Device identifier (physical)
  • SM identifier (physical)
  • Warp identifier (physical)
  • Lane identifier (physical)

• Divergent thread behavior
  • Consider: two or more threads in the same warp execute different instructions
    • Example: if else body
  • Active lane mask
    • Threads that are currently executing device code at $pc
  • Divergent lane mask
    • Threads that are waiting or have completed at $pc

[Switching focus to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device 0, sm 0, warp 0, lane 0]

• Kernel identifier (logical)
  • Assigned sequentially by CUDA-GDB
  • Unique across devices
  • Begins at index 0

• Grid identifier (logical)
  • Assigned by CUDA
  • Unique per device
  • Begins at index 1
  • CUDA dynamic parallelism can have negative grid offsets

• Block identifier (logical)
Overview: CUDA-GDB

Device information

• Use `info cuda` commands to query CUDA enabled GPU activities

```
(cuda-gdb) help info cuda
Print information about the current CUDA activities. Available options:
  devices : information about all the devices
  sms : information about all the SMs in the current device
  warps : information about all the warps in the current SM
  lanes : information about all the lanes in the current warp
  kernels : information about all the active kernels
  contexts : information about all the contexts
  blocks : information about all the active blocks in the current kernel
  threads : information about all the active threads in the current kernel
  launch trace : information about the parent kernels of the kernel in focus
  launch children : information about the kernels launched by the kernels in focus
  managed : information about global managed variables
  line : information about the filename and linenumber for a given $pc
```

• Output from `info cuda` marked with a * indicates that the range contains the focused CUDA thread
Overview: CUDA-GDB

Device information

- **info cuda kernels**
  - Displays the list of kernels

```
(cuda-gdb) info cuda kernels
     Kernel Parent Dev Grid Status       SMs Mask    GridDim BlockDim Invocation
        *         0  -  0    1 Active 0xfffffffffffff (20,10,1) (32,32,1) MatrixMulCUDA<32>()
```

- **info cuda blocks**
  - Displays the list of active blocks in the focused kernel

```
(cuda-gdb) info cuda blocks
     BlockIdx To BlockIdx Count State
Kernel 0
        *         0 (0,2,0) 3 running
        *         1 (1,2,0) 3 running
        *         2 (2,2,0) 3 running
        *         3 (3,2,0) 3 running
```
Overview: CUDA-GDB

Device information

- **info cuda threads**
  - Displays the active threads in the focused kernel

```
(cuda-gdb) info cuda threads
BlockIdx ThreadIdx To BlockIdx To ThreadIdx Count Virtual PC Filename Line
Kernel 0
  (0,0,0) (0,2,0) (31,31,0) 3072 0x00007fffff85e230 matrixMul.cu 62
  *(1,0,0) (1,2,0) (31,31,0) 3072 0x00007fffff85e230 matrixMul.cu 62
  (2,0,0) (2,2,0) (31,31,0) 3072 0x00007fffff85e230 matrixMul.cu 62
```

- Obtain current focus with cuda commands

```
(cuda-gdb) cuda kernel block thread
kernel 0, block (1,0,0), thread (3,0,0)
```
Overview: CUDA-GDB
CUDA thread focus

• CUDA thread focus is controlled with *cuda* commands
  • Sets focus to single CUDA thread
  • Some commands apply only to thread in focus
    • Printing local or shared variables
    • Printing registers
    • Printing stack contents

• Examples
  • Set focus to specified CUDA thread

(cuad-gdb) cuda thread 5
[Switching focus to CUDA kernel 0, grid 1, block (2,0,0), thread (5,0,0), device 0, sm 4, warp 0, lane 5]

• Set focus based on block and thread

(cuad-gdb) cuda block 2 thread 6
[Switching focus to CUDA kernel 0, grid 1, block (2,0,0), thread (6,0,0), device 0, sm 4, warp 0, lane 6]

• Set focus based on kernel, dim3 block, dim3 thread

(cuad-gdb) cuda kernel 0 block 1,0,0 thread 3,0,0
[Switching focus to CUDA kernel 0, grid 1, block (1,0,0), thread (3,0,0), device 0, sm 2, warp 0, lane 3]
Overview: CUDA-GDB
Execution Control Basics

• Two ways to get control
  • run
    $ cuda-gdb --quiet my_application
    Reading symbols from my_application...
    (cuda-gdb) run
  • attach
    $ cuda-gdb --quiet
    (cuda-gdb) attach 261230

• Exit debugger with quit
  • Applications run are killed
  • Applications attach are detached

• Resume application execution
  (cuda-gdb) continue
  • Resumes both host and device threads

• Interrupt execution with ctrl-c
  • Application is executing
  • No (cuda-gdb) prompt
  • Ctrl-C halts both host and device threads
Overview: CUDA-GDB

Stepping

• Single stepping
  • Source vs assembly level
  • Over vs into function calls
  • Device behavior is like host
    • Source level – following source line in kernel
    • Assembly level – following SASS instruction

<table>
<thead>
<tr>
<th>Stepping mode</th>
<th>Source level command</th>
<th>Assembly level command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Over functions</td>
<td>next</td>
<td>nexti</td>
</tr>
<tr>
<td>Into functions</td>
<td>step</td>
<td>stepi</td>
</tr>
</tbody>
</table>

• Stepping behaviors
  • Single stepping advances every active thread in the warp
  • Divergent inactive threads do not make forward progress
  • Kernel launch is asynchronous
    • Cannot step into launched kernel from host code
    • Set a breakpoint or use break_on_launch

• Stepping behaviors (cont.)
  • Stepping over barriers
    • Example: __syncthreads()
    • Resumes execution of all warps executing the same block
    • Required to make forward progress past barrier
Overview: CUDA-GDB

Breakpoints

- Symbolic breakpoints
  
  \[
  \text{(cuda-gdb) break my\_function} \\
  \text{(cuda-gdb) break my\_class::my\_method}
  \]

- Line breakpoints
  
  \[
  \text{(cuda-gdb) break my\_file.cu:185}
  \]

- Address breakpoints
  
  \[
  \text{(cuda-gdb) break *0x1afe34d0}
  \]

- Conditional breakpoints
  
  - Executed on the host every time breakpoint is hit
  - Can be slow
  
  \[
  \text{(cuda-gdb) break foo.cu:23 if threadIdx.x == 1 && i < 5}
  \]

- Kernel entry breakpoints
  
  - Used to automatically break on kernel launches
  - Good first step if you don't know where to start
  
  \[
  \text{(cuda-gdb) set cuda break\_on\_launch application}
  \]
Overview: CUDA-GDB

Breakpoints

- `info break`
  - View the status of breakpoints
  - Breakpoints can be pending
  - Breakpoints can be set at multiple addresses
  - Breakpoint locations may change during runtime

(cuda-gdb) break main
Breakpoint 1 at 0xbdaa: file matrixMul.cu, line 296.
(cuda-gdb) info break
Num | Type    | Disp | Enb | Address                  | What
--- | ------- |------|-----|--------------------------|-----
1   | breakpoint | keep | y   | 0x000000000000bdaa in main(int, char**) at matrixMul.cu:296

- Breakpoint resolution
  - Breakpoints inserted as pending until CUDA cubins are loaded
    - Missing most CUDA symbols
    - Host side shadow breakpoints can be inserted on named kernel
    - Automatically resolved to device location after cubin load
    - Missing line info
  - Similar debugging experience to `dlopen`
  - C++ templates may result in multiple breakpoint locations
## Overview: CUDA-GDB

### Breakpoints

- **Pending breakpoint examples**

```bash
(cuda-gdb) break MatrixMulCUDA
Breakpoint 2 at 0x555555561535: MatrixMulCUDA. (2 locations)
```

```bash
(cuda-gdb) info break 2
```

<table>
<thead>
<tr>
<th>Num</th>
<th>Type</th>
<th>Disp</th>
<th>Enb</th>
<th>Address</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>breakpoint</td>
<td>keep</td>
<td>y</td>
<td>&lt;MULTIPLE&gt;</td>
<td>MatrixMulCUDA&lt;16&gt;(float*, float*, float*, int, int) at matrixMul.cu:60</td>
</tr>
<tr>
<td>2.1</td>
<td></td>
<td></td>
<td>n</td>
<td>0x0000555555561535</td>
<td>MatrixMulCUDA&lt;16&gt;(float*, float*, float*, int, int) at matrixMul.cu:60</td>
</tr>
<tr>
<td>2.2</td>
<td></td>
<td></td>
<td>n</td>
<td>0x0000555555561576</td>
<td>MatrixMulCUDA&lt;32&gt;(float*, float*, float*, int, int) at matrixMul.cu:60</td>
</tr>
</tbody>
</table>

```bash
(cuda-gdb) info break 2
```

<table>
<thead>
<tr>
<th>Num</th>
<th>Type</th>
<th>Disp</th>
<th>Enb</th>
<th>Address</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>breakpoint</td>
<td>keep</td>
<td>y</td>
<td>&lt;MULTIPLE&gt;</td>
<td>MatrixMulCUDA&lt;16&gt;(float*, float*, float*, int, int) at matrixMul.cu:60</td>
</tr>
<tr>
<td>2.1</td>
<td></td>
<td></td>
<td>n</td>
<td>0x0000555555561535</td>
<td>MatrixMulCUDA&lt;16&gt;(float*, float*, float*, int, int) at matrixMul.cu:60</td>
</tr>
<tr>
<td>2.2</td>
<td></td>
<td></td>
<td>n</td>
<td>0x0000555555561576</td>
<td>MatrixMulCUDA&lt;32&gt;(float*, float*, float*, int, int) at matrixMul.cu:60</td>
</tr>
<tr>
<td>2.3</td>
<td></td>
<td></td>
<td>y</td>
<td>0x00007fffec85c130</td>
<td>MatrixMulCUDA&lt;16&gt;(float*, float*, float*, int, int) at matrixMul.cu:62</td>
</tr>
<tr>
<td>2.4</td>
<td></td>
<td></td>
<td>y</td>
<td>0x00007fffec85e230</td>
<td>MatrixMulCUDA&lt;32&gt;(float*, float*, float*, int, int) at matrixMul.cu:62</td>
</tr>
</tbody>
</table>
```
Overview: CUDA-GDB

Breakpoints

- Pending breakpoint examples (cont.)

```
(cuda-gdb) break matrixMul.cu:104
Breakpoint 3 at 0x555555561554: matrixMul.cu:104. (2 locations)
(cuda-gdb) info break 3
Num Type Disp Enb Address What
 3 breakpoint keep y <MULTIPLE>
 3.1 n 0x0000555555561554 in MatrixMulCUDA<16>(float*, float*, float*, int, int) at matrixMul.cu:129
 3.2 n 0x0000555555561595 in MatrixMulCUDA<32>(float*, float*, float*, int, int) at matrixMul.cu:129
```

```
(cuda-gdb) info break 3
Num Type Disp Enb Address What
 3 breakpoint keep y <MULTIPLE>
 3.1 y 0x00007fff385c4a0 in MatrixMulCUDA<16>(float*, float*, float*, int, int) at matrixMul.cu:104
 3.2 y 0x00007fff385e5a0 in MatrixMulCUDA<32>(float*, float*, float*, int, int) at matrixMul.cu:104
```
Overview: CUDA-GDB

Stacktrace

- Same commands as used in gdb
  - `where`, `bt`, `info stack`
- Applies to the thread in focus
- CUDA threads have first source line of kernel as outermost frame

```
(cuda-gdb) bt
#0  recursive_function (i=1) at calldepth_function.cu:4
#1  0x00007fffec85b690 in recursive_function (i=2) at calldepth_function.cu:7
#2  0x00007fffec85b690 in recursive_function (i=3) at calldepth_function.cu:7
#3  0x00007fffec85a890 in calldepth<<<(1,1,1),(2,1,1)>> (input=3, output=0x7ffec1e00000) at calldepth_kernel.cu:7
```
Overview: CUDA-GDB
Examining state

- info locals
  - Displays local variables in the current stack frame
  - Value printed or hint as to why the variable is not valid

```c
(cuda-gdb) info locals
by = <unavailable>
tx = <unavailable>
aStep = <unavailable>
bx = <unavailable>
ty = <unavailable>
aBegin = <unavailable>
aEnd = <unavailable>
bBegin = <unavailable>
bStep = <unavailable>
Csub = <optimized out>
c = <unavailable>
```

```c
by = blockIdx.y;
tx = threadIdx.x;
ty = threadIdx.y;
aBegin = wA * BLOCK_SIZE * by;
aEnd = aBegin + wA - 1;
aStep = BLOCK_SIZE;
bBegin = BLOCK_SIZE * bx;
```

```c
(by = 0
tx = 0
aStep = 32
bx = 0
ty = 0
aBegin = 0
aEnd = 319
bBegin = 32
bStep = <unavailable>
Csub = <optimized out>
c = <unavailable>
```
Overview: CUDA-GDB
Examining state

- **print**
  - Read a source variable
  - Variable must be in scope
    - Local or global scope

```cuda-gdb
(print A[1]
$1 = 1
(print &A[1]
$2 = (@generic float *) 0x7fffc3a00004
```

- **set variable**
  - Write to a source variable
  - Address space must have write permissions

```cuda-gdb
(print bx
$3 = 0
(set variable bx = 3
(print bx
$4 = 3
```

- Supply address space identifier when storage class is ambiguous
  - @code, @constant, @generic, @global, @managed_global, @parameter, @shared, @register, @local, @uniform_register

- **info registers**
  - Inspect device registers
  - Pseudo names
    - $R<num>
      - Regular register
    - $UR<num>
      - Uniform register
    - $UP<num>
      - Uniform predicate
    - $PC
      - Program counter
    - Unassignable
Overview: CUDA-GDB

API Errors

- set cuda api_failures
  - Allows automatic checks of any CUDA driver or runtime API call
  - Three modes
    - hide
      - Do not report error of any kind
    - ignore
      - Emit warning, but continue execution
      - Default
    - stop
      - Emit an error and stop execution

```
(cuda-gdb) set cuda api_failures stop
(cuda-gdb) continue
Continuing.
Cuda API error detected: cudaMalloc returned (0x1)
(cuda-gdb)
```
Overview: CUDA-GDB

GPU Exceptions

- GPU device exceptions
  - Always caught
  - Fatal – unable to continue device execution
  - Most exceptions are precise
    - View address causing exception with `errorpc`
    - CUDA cluster (CUDA 11.8+) exceptions are imprecise
      - Use `autostep` to determine exact block and instruction causing error
  - CUDA_EXCEPTION_0 through CUDA_EXCEPTION_18
    - See link for table of exceptions and descriptions: [https://docs.nvidia.com/cuda/cuda-gdb/index.html#gpu-error-reporting](https://docs.nvidia.com/cuda/cuda-gdb/index.html#gpu-error-reporting)
Overview: CUDA-GDB

GPU Exceptions

- Table of exception codes

<table>
<thead>
<tr>
<th>Exception Code</th>
<th>Precision of the Error</th>
<th>Scope of the Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA_EXCEPTION_0</td>
<td>“Device Unknown Exception”</td>
<td>Unknown</td>
<td>Global error on the GPU</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_1</td>
<td>“Deprecated”</td>
<td>Deprecated</td>
<td>Deprecated</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_2</td>
<td>“Lane User Stack Overflow”</td>
<td>Precise</td>
<td>Per lane/thread error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_3</td>
<td>“Device Hardware Stack Overflow”</td>
<td>Precise</td>
<td>Global error on the GPU</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_4</td>
<td>“Warp Illegal Instruction”</td>
<td>Precise</td>
<td>Warp error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_5</td>
<td>“Warp Out-of-range Address”</td>
<td>Precise</td>
<td>Warp error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_6</td>
<td>“Warp Misligned Address”</td>
<td>Precise</td>
<td>Warp error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_7</td>
<td>“Warp Invalid Address Space”</td>
<td>Precise</td>
<td>Warp error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_8</td>
<td>“Warp Invalid PC”</td>
<td>Precise</td>
<td>Warp error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_9</td>
<td>“Warp Hardware Stack Overflow”</td>
<td>Precise</td>
<td>Warp error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_10</td>
<td>“Device Illegal Address”</td>
<td>Precise</td>
<td>Global error</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_11</td>
<td>“Deprecated”</td>
<td>Deprecated</td>
<td>Deprecated</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_12</td>
<td>“Warp Assert”</td>
<td>Precise</td>
<td>Per warp</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_13</td>
<td>“Deprecated”</td>
<td>Deprecated</td>
<td>Deprecated</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_14</td>
<td>“Warp Illegal Address”</td>
<td>Precise</td>
<td>Per warp</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_15</td>
<td>“Invalid Managed Memory Access”</td>
<td>Precise</td>
<td>Per host thread</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_16</td>
<td>“Deprecated”</td>
<td>Deprecated</td>
<td>Deprecated</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_17</td>
<td>“Cluster Out-of-range Address”</td>
<td>Not precise</td>
<td>Per Cuda Cluster</td>
</tr>
<tr>
<td>CUDA_EXCEPTION_18</td>
<td>“Cluster Target Block Not Present”</td>
<td>Not precise</td>
<td>Per Cuda Cluster</td>
</tr>
</tbody>
</table>
Overview: CUDA-GDB

GPU Exceptions

- GPU exception example

CUDA Exception: Warp Out-of-range Address
The exception was triggered at PC 0x7fffc385acd0 (memexceptions_kernel.cu:21)

Thread 1 "memexceptions" received signal CUDA_EXCEPTION_5, Warp Out-of-range Address.
[Switching focus to CUDA kernel 1, grid 1, block (0,0,0), thread (0,0,0), device 0, sm 0, warp 0, lane 0]
exception_kernel<<<(1,1,1),(1,1,1)>>> (data=0x7fffc1e00000, exception=OOR_SHARED) at memexceptions_kernel.cu:44
44
   *sdata = *ldata;

(cuda-gdb) print $errorpc
$1 = (void (*)(void)) 0x7fffc385acd0 <exception_kernel(void*, exception_t)+1488>

(cuda-gdb) print $pc
$2 = (void (*)(void)) 0x7fffc385b1d0 <exception_kernel(void*, exception_t)+2768>

(cuda-gdb) list *$errorpc
0x7fffc385acd0 is in exception_kernel(void*, exception_t) (memexceptions_kernel.cu:21).
16   case MMU_FAULT:
17       *(volatile unsigned char *)0 = exception;
18       // Above line causes an MMU fault (global page not mapped for writing)
19           break;
20   case OOR_SHARED:
21       *(volatile unsigned char *)(sdata + gridDim.x*MAX_SHARED) = exception;
22       // Above line causes an out-of-range access (shared)
23           break;
24   case OOR_LOCAL:
25       *(volatile unsigned char *)(ldata + gridDim.x*MAX_LOCAL) = exception;
Overview: CUDA-GDB

Disassembly

- disassemble
  - View disassembly of sass instructions
  - Current pc prefixed with =>
  - Instruction triggering exception (errorpc) prefixed with »>
    - If errorpc and pc match, prefixed with *=>

(cuda-gdb) disas $pc,+32
Dump of assembler code from 0x7fffc385b4b0 to 0x7fffc385b4d0:
=>0x00007fffc385b4b0 <_Z16exception_kernelPv11exception_t+3504>: ERRBAR
 0x00007fffc385b4c0 <_Z16exception_kernelPv11exception_t+3520>: EXIT
End of assembler dump.

(cuda-gdb) disas $errorpc,+64
Dump of assembler code from 0x7fffc385ab20 to 0x7fffc385ab60:
*> 0x00007fffc385ab20 <_Z16exception_kernelPv11exception_t+1056>: ST.E.U8.STRONG.SYS desc[UR4][R6.64], R5
 0x00007fffc385ab30 <_Z16exception_kernelPv11exception_t+1072>: BRA 0xad0
 0x00007fffc385ab40 <_Z16exception_kernelPv11exception_t+1088>: PRMT R5, R5, 0x7610, R5
 0x00007fffc385ab50 <_Z16exception_kernelPv11exception_t+1104>: MOV R6, c[0x0][0xc]
End of assembler dump.
Overview: CUDA-GDB

Coredumps

- GPU coredump support
  - Disabled by default
  - Set CUDA_ENABLE_COREDUMP_ON_EXCEPTION env var to 1
  - Generated when a GPU exception is encountered

$ ./memexceptions 1
SM version: 86, Min version: 35, Max version: 999
Aborted (core dumped)
$ ls | grep core
core_1669651659_agontarek-dt_612954.nvcudmp

- GPU coredump name
  - core_%t_%h_%p.nvcudmp
    - %t is seconds since Epoch
    - %h is hostname of system running the CUDA application
    - %p is the process identifier of the CUDA application
  - Written into the applications $PWD by default
  - User defined with CUDA_COREDUMP_FILE env var
    - Recognizes %t, %h, %p specifiers

$ export CUDA_COREDUMP_FILE="/lus/grand/projects/alcf_training/$USER/core.gpu.%h.%p"
Overview: CUDA-GDB
Coredumps

- Lightweight coredumps
  - Set `CUDA_ENABLE_LIGHTWEIGHT_COREDUMP` env var to 1
  - GPU coredumps will forego dumping memory
    - Local
    - Shared
    - Global
- Size of coredump reduced significantly
- Backtrace only
Overview: CUDA-GDB

Coredumps

• User induced GPU coredump
  • Set CUDA_ENABLE_USER_TRIGGERED_COREDUMP env var to 1
  • Opens a communication pipe for each CUDA process
  • Write to pipe to induce a GPU coredump

$ export CUDA_ENABLE_USER_TRIGGERED_COREDUMP=1
$ ./matrixMul > output &
  [1] 619157
$ ls | grep corepipe
corepipe_agontarek-dt_619157
$ echo "1" > corepipe_agontarek-dt_619157
  [1]+ Aborted                 (core dumped) ./matrixMul > output
$ ls | grep 619157
core_1669654018_agontarek-dt_619157.nvcudmp

• GPU corepipe name
  • corepipe_%h_%p
  • Same %t, %h, %p specifiers
  • User defined with CUDA_COREDUMPPIPE env var
Overview: CUDA-GDB

Coredumps

- **target cudacore**
  - Loads GPU core dump into the debugger
  - Can load both CPU and GPU coredumps
    - CPU coredump is optional
  - Examining coredumps with CUDA-GDB does not require a GPU be installed on the system

```bash
(cuda-gdb) target cudacore core_1669651659_agontarek-dt_612954.nvcudmp
Opening GPU coredump: core_1669651659_agontarek-dt_612954.nvcudmp

CUDA Exception: Warp Illegal Address
The exception was triggered at PC 0x7f2823859620 (memexceptions_kernel.cu:17)
[Current focus set to CUDA kernel 0, grid 1, block (0,0,0), thread (0,0,0), device 0, sm 0, warp 0, lane 0]
#0 0x00007f2823859fb0 in exception_kernel<<<(1,1,1),(1,1,1)>> (data=0x7f2820c00000, exception=MMU_FAULT) at memexceptions_kernel.cu:50
50}
(cuda-gdb) print $errorpc
$1 = (void (*)(void)) 0x7f2823859620 <exception_kernel(void*, exception_t)+1056>
(cuda-gdb) print $pc
$2 = (void (*)(void)) 0x7f2823859fb0 <exception_kernel(void*, exception_t)+3504>
```
Overview: Compute Sanitizer
Overview: Compute Sanitizer

What is it?

- Suite of dynamic analysis tools to catch common programming errors
  - Memcheck
    - Report invalid memory accesses
  - Initcheck
    - Report uninitialized memory reads
  - Racecheck
    - Report invalid concurrent accesses to shared memory
  - Synccheck
    - Report invalid barrier usage

- Non-interactive CLI based tool

- Provides proactive debugging of CUDA kernels
  - Discover common programming errors up front

- Supports CUDA/OptiX/OpenACC/OpenMP/etc

- Replaces CUDA-MEMCHECK tool
  - Deprecated since CUDA 11.5
  - Removed in next major version
  - CUDA-GDB memcheck support removed
    - Sanitizer coredumps

- Removed in next major version
Overview: Compute Sanitizer

Quickstart

- On Polaris
  - Missing from PATH by module nvhp/c21.9 (default)
    ```bash
    agontarek@polaris-login-01:~> $NVIDIA_PATH/cuda/11.4/compute-sanitizer/compute-sanitizer
    ```
  - Provided by path from module nvhp/c22.7 (non-default)
    ```bash
    agontarek@polaris-login-01:~> which compute-sanitizer
    ```
  ```bash
  /soft/ecp/ParaTools/E4S/22.08/mvapich2/spack/opt/spack/cray-sles15-zen3/gcc-11.2.0/nvhpc-22.7-bpsppgyo3xpqdbltyxlkyjyzbml57/Linux_x86_64/22.7 compilers/bin/compute-sanitizer
  ```
- Recompile for debugging
  - When compiling with nvcc:
    - Provide `-g` for host (CPU) debugging
    - Provide `-G` for device (GPU) debugging
  - Using `-lineinfo` will allow checking of optimized code
    - Reduced quality of output messages
- Latest documentation: [https://docs.nvidia.com/compute-sanitizer/index.html](https://docs.nvidia.com/compute-sanitizer/index.html)
- Compute sanitizer examples: [https://github.com/NVIDIA/compute-sanitizer-samples](https://github.com/NVIDIA/compute-sanitizer-samples)
Overview: Compute Sanitizer

Memcheck

• Memcheck is used to report invalid memory accesses
  • Out of bounds or misaligned read/write/atomic accesses
    • Local, shared, or global memory
  • Stack overflows
    • Invalid system-scoped atomic accesses
      • NVLINK peer access
• Reports CUDA API errors
• Hardware exceptions
• Invalid device-side malloc/free usage
• Default tool for compute-sanitizer
Overview: Compute Sanitizer

Memcheck

• When first error is encountered
  • Destroy the CUDA context by default
  • Controllable with args
    --destroy-on-device-error=<context|kernel>

---

```c
__device__ void writeIdx(int* buffer) {
    buffer[threadIdx.x] = threadIdx.x;
}

__global__ void kernel(int* buffer) {
    writeIdx(buffer);
}

int main() {
    void* devBuf = nullptr;
    cudaMalloc(&devBuf, 31 * sizeof(int));
    kernel<<<1,32>>>(static_cast<int*>(devBuf));
    return cudaDeviceSynchronize();
}
```

---

• When first error is encountered
  • Destroy the CUDA context by default
  • Controllable with args
    --destroy-on-device-error=<context|kernel>
Overview: Compute Sanitizer

Memcheck

- Report device side memory leaks
- --leak-check=full

```c
__device__ void writeIdx(int* buffer)
{
    buffer[threadIdx.x] = threadIdx.x;
}

__global__ void kernel(int* buffer)
{
    writeIdx(buffer);
}

int main()
{
    void* devBuf = nullptr;
    cudaMalloc(&devBuf, 31 * sizeof(int));
    kernel<<<1, 32>>>(static_cast<int*>(devBuf));
    return cudaDeviceSynchronize();
}
```
Overview: Compute Sanitizer

Memcheck

- Avoid false negative invalid memory accesses with padding
  - Adds a padding buffer at the end of each allocation
  - Ensures out-of-bounds access doesn’t access adjacent memory allocation
  - --padding=<bytes>
Overview: Compute Sanitizer

Initcheck

- Initcheck is used to report uninitialized memory reads
  - Kernel
  - Memory passed to CUDA API calls
- Global memory supported
  - Shared and local memory untracked
- Can track peer GPU allocations
Overview: Compute Sanitizer

Initcheck

```c
__global__ void kernel(int* buffer)
{
    buffer[threadIdx.x] = buffer[threadIdx.x] + threadIdx.x;
}

int main()
{
    void* devBuf = nullptr;
    cudaMalloc(&devBuf, 32 * sizeof(int));
    kernel<<<1,1>>>(static_cast<int*>(devBuf));
    return cudaDeviceSynchronize();
}
```

agontarek@x3204c0s13b0n0:~/.sanitizer_demos> compute-sanitizer --tool=initcheck ./initcheck-test

======== COMPUTE-SANITIZER

- Uninitialized __global__ memory read of size 4 bytes
- at 0x1a0 in /home/agontarek/sanitizer_demos/initcheck.cu:3:kernel(int *)
- by thread (0,0,0) in block (0,0,0)
- Address 0x152063000000
- Saved host backtrace up to driver entry point at kernel launch time
- Host Frame: [0x20d4ea] in /usr/lib64/libcuda.so.1
- Host Frame:__cudart802 [0x87ab] in /home/agontarek/sanitizer_demos/.initcheck-test
- Host Frame:cudaLaunchKernel [0x5ede8] in /home/agontarek/sanitizer_demos/.initcheck-test
- Host Frame:/opt/nvidia/hpc_sdk/Linux_x86_64/21.9/cuda/include/cuda_runtime.h:211:cudaError
- cudaLaunchKernel-char(char const*, dim3, dim3, void**, unsigned long, CUstream_st*) [0x4053]
- in /home/agontarek/sanitizer_demos/.initcheck-test
- Host Frame:/var/tmp/pbs.357595.polaris-pbs-01.0sn.cm.polaris.anl.gov/tmpxft_0000d4b6_00000000-6_initcheck.cudafe1.stub.c:13:_device_stub_Z4kernelPl(int*) [0x3f34]
- in /home/agontarek/sanitizer_demos/.initcheck-test
- Host Frame:/home/agontarek/sanitizer_demos/initcheck.cu:4:kernel(int*) [0x3f5c]
- in /home/agontarek/sanitizer_demos/.initcheck-test
- Host Frame:/home/agontarek/sanitizer_demos/initcheck.cu:11:main [0x3ede2]
- in /home/agontarek/sanitizer_demos/.initcheck-test
- Host Frame:...sysdeps/x86_64/start.S:122:_start [0x3e7a]
- in /home/agontarek/sanitizer_demos/.initcheck-test

======== ERROR SUMMARY: 1 error
Overview: Compute Sanitizer

Initcheck

- Initcheck can track unused memory
- Global memory allocated but never written
- --track-unused-memory=yes

```c
__global__ void kernel(int* buffer)
{
    buffer[threadIdx.x] = buffer[threadIdx.x] + threadIdx.x;
}

int main()
{
    void* devBuf = nullptr;
    cudaMalloc(&devBuf, 32 * sizeof(int));
    kernel<<<1,1>>>(static_cast<int*>(devBuf));
    return cudaDeviceSynchronize();
}
```
Overview: Compute Sanitizer

Racecheck

- Racecheck is used to detect potential race conditions
  - WAW, WAR, RAW accesses to shared memory
  - Lack of valid synchronization primitive
    - Warp/block level etc
  - Shared memory supported
    - Global and local memory untracked
- Two reporting modes
  - Analysis
    - Aggregated report
  - Hazard
    - Every detected error with details
    - Verbose
Overview: Compute Sanitizer

Racecheck

```c
__global__ void kernel(int* buffer)
{
    __shared__ int shared[64];

    shared[threadIdx.x] = threadIdx.x;
    buffer[threadIdx.x] = shared((threadIdx.x + 1) % 64);
}

int main()
{
    void* devBuf = nullptr;
    cudaMalloc(&devBuf, 64 * sizeof(int));
    kernel<<<64>>>(static_cast<int*>(devBuf));
    return cudaDeviceSynchronize();
}
```

agontarek@x3204c0s13b0n0:~/sanitizer_demos> compute-sanitizer --tool=racecheck ./racecheck-test

======== COMPUTE-SANITIZER
======== Error: Race reported between Write access at 0x250 in /home/agontarek/sanitizer_demos/racecheck.cu:5:kernel(int *)
        and Read access at 0x5d0 in /home/agontarek/sanitizer_demos/racecheck.cu:6:kernel(int *) [256 hazards]
======== RACECHECK SUMMARY: 1 hazard displayed (1 error, 0 warnings)

agontarek@x3204c0s13b0n0:~/sanitizer_demos> compute-sanitizer --tool=racecheck --racecheck-report=hazard ./racecheck-test | head -n 25

======== COMPUTE-SANITIZER
======== Warning: (Warp Level Programming) Potential RAW hazard detected at __shared__ 0x84 in block (0,0,0) :
        Write Thread (33,0,0) at 0x250 in /home/agontarek/sanitizer_demos/racecheck.cu:5:kernel(int *)
        Read Thread (32,0,0) at 0x5d0 in /home/agontarek/sanitizer_demos/racecheck.cu:6:kernel(int *)
        Current Value : 33
        Saved host backtrace up to driver entry point at kernel launch time
        Host Frame: [0x20d4ea]
        in /usr/lib64/libcuda.so.1
        Host Frame:__cudart802 [0x87ab]
        in /home/agontarek/sanitizer_demos./racecheck-test
        Host Frame:cudaLaunchKernel [0x5ed66]
        in /home/agontarek/sanitizer_demos./racecheck-test
        Host Frame:/opt/nvidia/hpc_sdk/Linux_x86_64/21.9/cuda/11.4/include/cuda_runtime.h:211:cudaError cudaLaunchKernel<char>(char
        const*, dim3, dim3, void**, unsigned long, CUDStream_st*) [0x4053]
        in /home/agontarek/sanitizer_demos./racecheck-test
        Host Frame:/var/tmp/pbs.357595.polaris-pbs-01.hsn.cn.polaris.alcf.anl.gov/tmpxft_0000d332_00000000-6_racecheck.cudafe1.stub.c:13:__device_stub__Z6kernelPl(int*) [0x3f34]
        in /home/agontarek/sanitizer_demos./racecheck-test
        Host Frame:/home/agontarek/sanitizer_demos/racecheck.cu:7:kernel(int*) [0x3f5c]
        in /home/agontarek/sanitizer_demos./racecheck-test
        Host Frame:/home/agontarek/sanitizer_demos/racecheck.cu:14:main [0x3de2]
        in /home/agontarek/sanitizer_demos./racecheck-test
        Host Frame:__libc_start_main [0x2534d]
        in /lib64/libc.so.6
        Host Frame:sysdeps/x86_64/start.S:122:_start [0x3c7a]
        in /home/agontarek/sanitizer_demos./racecheck-test
```
Overview: Compute Sanitizer

Synccheck

- Synccheck is used to detect invalid use of CUDA synchronization primitives
- Behavior depends on architecture
  - Divergent threads in warp/block
  - Invalid barrier arguments
Overview: Compute Sanitizer

Synccheck

```c
#include <cuda/barrier>

__global__ void kernel()
{
    __shared__ cuda::barrier<cuda::thread_scope_block> barrier;

    if (threadIdx.x == 0)
    {
        init(&barrier, blockDim.x / 2);
    }

    __syncthreads();

    auto token = barrier.arrive();
    barrier.wait(std::move(token));
}

int main()
{
    kernel<<<1,32>>>();
    return cudaDeviceSynchronize();
}
```
Overview: Compute Sanitizer

Synccheck

agontarek@x3204c0s13b0n0:/sanitizer_demos> compute-sanitizer --tool=synccheck ./synccheck-test
======== COMPUTE-SANITIZER
======== Barrier error detected. Barrier overflow
======== at 0x540 in
/opt/nvidia/hpc_sdk/Linux_x86_64/21.9/cuda/11.4/include/cuda/std/barrier:189:cuda::_3::barrier<(cuda::_3::thread_scope)2,
cuda::std::_3::__empty_completion>::arrive(long)
======== by thread (31,0,0) in block (0,0,0)
======== Device Frame:/home/agontarek/sanitizer_demos/synccheck.cu:14:kernel() [0x6f0]
======== Saved host backtrace up to driver entry point at kernel launch time
======== Host Frame: [0x20d4ea]
in /usr/lib64/libcuda.so.1
Host Frame:__cudart802 [0x876b]
in /home/agontarek/sanitizer_demos./synccheck-test
Host Frame:cudaLaunchKernel [0x5eda8]
in /home/agontarek/sanitizer_demos./synccheck-test
Host Frame:/opt/nvidia/hpc_sdk/Linux_x86_64/21.9/cuda/11.4/include/cuda_runtime.h:211:cudaError cudaLaunchKernel<char>(char
const*, dim3, dim3, void**, unsigned long, CUstream_st*) [0x400e]
in /home/agontarek/sanitizer_demos./synccheck-test
Host Frame:/var/tmp/pbs.357595.polaris-pbs-01.hsn.cm.polaris.alcf.anl.gov/tmpxft_0000d43f_00000000-6_synccheck.cudafe1.stub.c:13:_device_stub_Z6kernelv() [0x3ec]
in /home/agontarek/sanitizer_demos./synccheck-test
Host Frame:/home/agontarek/sanitizer_demos/synccheck.cu:16:kernel() [0x3f17]
in /home/agontarek/sanitizer_demos./synccheck-test
Host Frame:/home/agontarek/sanitizer_demos/synccheck.cu:21:main [0x3dc2]
in /home/agontarek/sanitizer_demos./synccheck-test
Host Frame:__libc_start_main [0x2534d]
in /lib64/libc.so.6
Host Frame:../sysdeps/x86_64/start.S:122:_start [0x3c7a]
in /home/agontarek/sanitizer_demos./synccheck-test
========
Target application returned an error
======== ERROR SUMMARY: 1 error
Overview: Compute Sanitizer

Useful options

• Track all child processes
  • --target-processes=all

• Filter desired kernel launches to be tracked
  • --kernel-regex
  • --kernel-regex-exclude

• Track/ignore n kernel launches
  • --launch-count=n
  • --launch-skip=n

• Force stream synchronization every n launches
  • --force-synchronization-limit

• XML output for error reports
  • --xml=yes

• Generate coredump on first error
  • --generate-coredump=yes

• Debug with CUDA-GDB
  • Unsupported with racecheck

• Support for custom memory allocators with NVIDIA Tools Extension (NVTX)
Questions/Comments?