CUDA-GDB

NVIDIA CUDA Debugger

What is CUDA-GDB?

- Command-line debugger
- GDB extension, open-sourced (GPL)
- Linux (GDB 6.6) and MAC (GDB 6.3.5)
- 32-bit and 64-bit applications
- C and C++ (v4.0) applications
- Simultaneously debug host and device code
- No OpenCL debugging

What does CUDA-GDB do?

- Control the execution of the application
  - Breakpoints
  - Single-step
  - CTRL-C
- Inspect the current state of the application
  - Kernels, blocks, threads
  - Devices, SMs, warps
- Inspect and Modify
  - Code memory (disassemble)
  - Global, shared, and local memory
  - Hardware registers
  - Textures (read-only)

New features in 4.0

- Mac OS
- C++ debugging
- Fermi disassembly
- Automatic breakpoints on kernel entries
- Conditional breakpoints
- Texture access
- Debugging kernels with textures
- Three-dimensional grid support

Installation and Usage

- Part of the CUDA Toolkit, available at
  - http://www.nvidia.com/object/cuda_get.html
- Add cuda-gdb to your $PATH
  - export PATH=/usr/local/cuda/bin:$PATH
- Compile the application
  - Include debug information with nvcc –g (host) and -G (dev)
  - Include statically compiled kernels for Tesla
    - -gencode arch=compute_10, code=sm_10
  - Include statically compiled kernels for Fermi
    - -gencode arch=compute_20, code=sm_20
- Invoke the debugger
  - cuda-gdb my_application

Usage Scenarios

- Restriction
  - Desktop manager and application cannot share GPUs
  - Otherwise, hitting a breakpoint would freeze the desktop manager
- Single-GPU systems
  - console mode only
- Multi-GPU systems
  - without desktop manager (console mode)
    - all GPUs visible to the application
  - with desktop manager
    - Linux: device used by X11 is hidden from the application
    - MAC: device used by Aqua must be manually hidden:
      - CUDA_VISIBLE_DEVICES=0,1,…

Workshop 1

Workshop 1

```
$ cuda-gdb -q vectorAdd
Using host libthread_db library “…”

(cuda-gdb) break vectorAdd
Breakpoint 1 at 0x400fa0: file vectorAdd.cu, line 4

(cuda-gdb) run
Starting program: …/workshop1/vectorAdd
[Thread debugging using libthread_db enabled]
[New process 17091]
[New thread …]
[Launch of CUDA Kernel 0 (vectorAdd) on Device 0]
[Switching to CUDA Kernel 0 (<<<(0,0),(0,0,0)>>>)]

Breakpoint 1, vectorAdd<<<(2,1), (512, 1, 1)>>> (A=0x…, B=0x…, C=0x…) at vectorAdd.cu: 5

5           int tid = threadIdx.x;
```

Workshop 1

```
(cuda-gdb) next
6               C[tid] = A[tid] + B[tid];

(cuda-gdb) info cuda threads
<<<(0,0),(0,0,0)>>> … <<<(0,0),(31,0,0)>>> vectorAdd<<<(2,1), (512, 1, 1)>>> (A=0x…, B=0x…, C=0x…) at vectorAdd.cu:6
<<<(0,0),(32,0,0)>>> … <<<(1,0),(511,0,0)>>> vectorAdd<<<(2,1), (512, 1, 1)>>> (A=0x…, B=0x…, C=0x…) at vectorAdd.cu:5

(cuda-gdb) info locals
tid = 0
A = (@global int * @parameter) 0x100000
B = (@global int * @parameter) 0x101000
C = (@global int * @parameter) 0x102000
```

Workshop 1

```
(cuda-gdb) print tid
$1 = 0

(cuda-gdb) continue
Continuing.
[Termination of CUDA Kernel 0 (vectorAdd) on Device 0]

Program exited normally
```

GDB Command Extension Philosophy

- Command behaves the same on device and host
  - Reuse existing GDB commands
  - Examples: info stack, break, …
- Command is new or behaves differently on device
  - new command
  - use the cuda prefix
  - Example: info cuda threads
- Command-line help
  - use the help command
  - Examples: help info cuda, help cuda, help set cuda

Execution Control

- Execution control is identical to host debugging:
  - launch the application
    - **(cuda-gdb) run [arguments]**
  - resume the application (all host and dev threads)
    - **(cuda-gdb) continue**
  - kill the application
    - **(cuda-gdb) kill**
  - interrupt the application
    - CTRL-C
  - single-step warp(s)

Single-stepping	At the source level	At the assembly level
Over function calls	next	nexti
Into function calls	step	stepi

Single-Stepping Scope

- The behavior of single-stepping depends on the presence of a thread synchronization instruction

PC at a barrier?	Single-stepping applies to	Notes
Yes	Warp in focus and all the warps that are running the same block	Required to step over the barrier
No	Warp in focus only	

Breakpoints

- Symbolic breakpoints
  - **(cuda-gdb) break my_kernel**
  - **(cuda-gdb) break _Z6kernelIfiEvPT_PT0_**
  - **(cuda-gdb) break int function<int>(int)**
- Line number breakpoints
  - will create multiple breakpoints if inside template functions
  - **(cuda-gdb) break my_app.cu:380**
- Address breakpoints
  - **(cuda-gdb) break *0x3e840a8**
  - **(cuda-gdb) break *$pc**
- Kernel entry breakpoints
  - **(cuda-gdb) set cuda break_on_launch application**
- List of breakpoints
  - **(cuda-gdb) info breakpoints**

Conditional Breakpoints (v4.0)

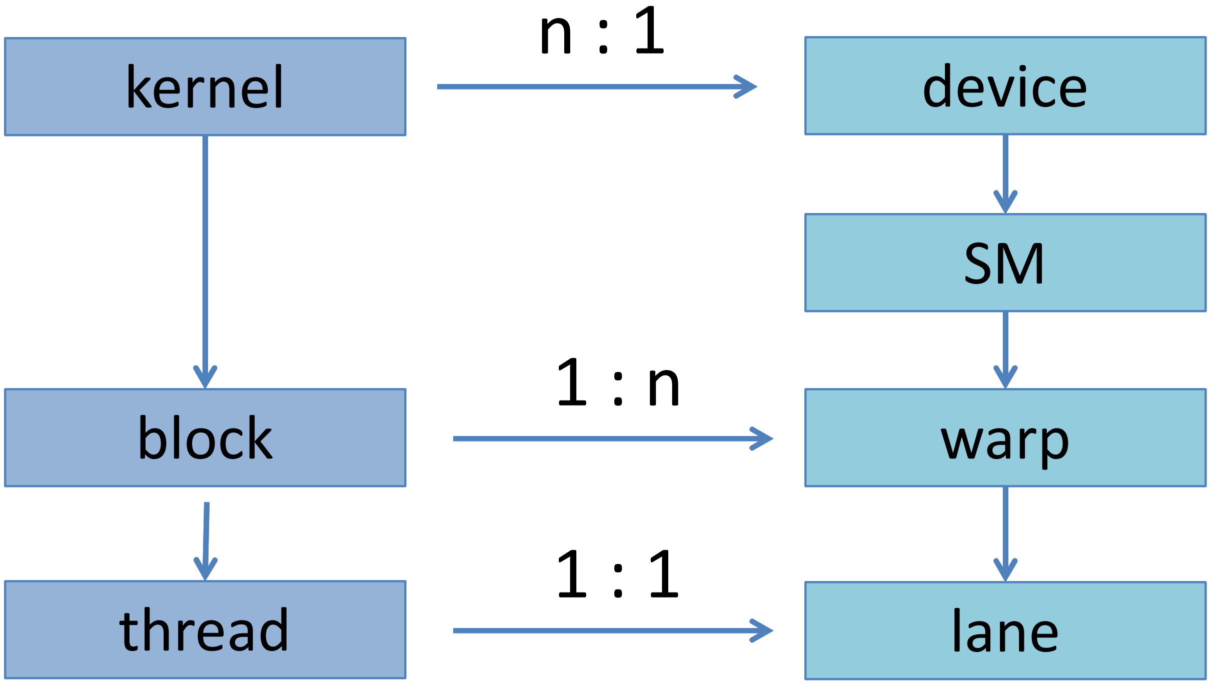
- Only reports hit breakpoints if the condition is met
  - all the breakpoints are still hit
  - condition is evaluated every time for all the threads
  - may slow down execution
- Set at breakpoint creation time
  - **(cuda-gdb) break my_kernel if threadIdx.x == 13**
- Set after the breakpoint is created (1 is the breakpoint number)
  - **(cuda-gdb) condition 1 blockIdx.x == 0 && n > 3**

Focus

- Many threads to deal with
  - how to decide which one the user wants?
- Concept of current focus
  - either host thread or device thread
  - which kernel/block/thread the user is looking at
  - cuda-gdb commands apply to the current focus
- Two different views for a device thread
  - hardware coordinates: device, SM, warp, lane
  - software coordinates: kernel, block, thread
  - mapping between the two sets of coordinates

Mapping Between Software and Hardware Coordinates

- a device may execute multiple kernels
- a block may run on multiple warps



kernel

n : 1

device

SM

block

1 : n

warp

thread

1 : 1

lane

Focus Query

- Query commands
  - cuda <list of coordinates>
  - thread
- If focus set to device thread
  - **(cuda-gdb) cuda kernel block thread**
    - kernel 1, block (0, 0, 0), thread (0, 0, 0)
  - **(cuda-gdb) cuda device kernel block warp thread**
    - kernel 1, block (0, 0, 0), thread (0, 0 ,0), device 0,warp 0
- If focus set to host thread
  - **(cuda-gdb) thread**
    - [Current thread is 1 …]
  - **(cuda-gdb) cuda thread**
    - Focus not set on any active CUDA kernel

Focus Switch

- Switch command
  - cuda <list of coordinate-value pairs>
  - thread <host thread id>
- Only switch the specified coordinates
  - current coordinates are assumed in case of non-specified coordinates
  - if no current coordinates, best effort to match request
  - error if cannot match request

Focus Switch

- **(cuda-gdb) cuda kernel 1 block 1 thread 2,0**
  - [Switching focus to CUDA kernel 1, grid 2, block (1, 0, 0), thread (2, 0, 0), device 0, sm 5, warp 0, lane 2]
- **(cuda-gdb) cuda block (1, 0, 0) lane 7 sm 5**
  - [Switching focus to CUDA kernel 1, grid 2, block (1, 0, 0), thread (7, 0, 0), device 0, sm 5, warp 0, lane 7]
- **(cuda-gdb) cuda kernel 1**
  - [Switching focus to CUDA kernel 1, grid 2, block (0, 0, 0), thread (0, 0, 0), device 0, sm 1, warp 0, lane 0]
- **(cuda-gdb) cuda thread 256**
  - Request cannot be satisfied. CUDA focus unchanged.

Workshop 2

Workshop 2

```
$ cuda-gdb -q matrixMul
(cuda-gdb) break matrixMul
[Breakpoint 1 at …]

(cuda-gdb) run
…
[Switching to CUDA kernel 0]
Breakpoint 1, matrixMul<<<(5,6),(16,16,1)>>>
4      int bx = blockIdx.x;

(cuda-gdb) cuda thread
[Current CUDA kernel 0 (thread (0,0,0))]
```

Workshop 2

```
(cuda-gdb) info cuda threads
<<<(0,0)(0,0,0)>>> … <<<(4,5)(15,15,0)>>>matrixMul

(cuda-gdb) next

(cuda-gdb) info cuda threads

(cuda-gdb) cuda thread (0,2,0)
[Switching to CUDA kernel 0 (device 0, …, thread(0,2,0)

(cuda-gdb) cuda block (0,1)
```

Program State Inspection (Terminology)

- PC (program counter)
  - virtual PC
    - address in the host virtual address space
    - always use virtual PC in cuda-gdb commands
  - physical PC
    - physical offset from the kernel entry point
    - useful when comparing to cuobjdump ouput
- Divergence
  - if 2 threads on the same warp must execute different instructions, the other must wait
  - active lanes: lanes currently executing device code
  - divergent lanes: lanes that are waiting for their turn or are done with their turn

Stack Trace

- Same (aliased) commands as in gdb:
  - where, backtrace, info stack
- Device stack trace detached from host stack trace
  - because the kernel launches are asynchronous
- Applies to the thread in focus
- Example
  - **(cuda-gdb) info stack**
    - #0 function<int> (t = 3)a t foo.cu:7
    - #1 0x0910a868 in kernel<int,float><<<(1,1,1),(1,1,1)>>>(out=0x2) at foo.cu:18
- On Tesla, all the functions are always inlined

State of the Application

- gdb command to get information about a topic:
  - **(cuda-gdb) info <topic>**
- cuda-gdb command to get information about a CUDA topic:
  - **(cuda-gdb) info cuda <topic>**
- info cuda topics:
  - kernels
  - blocks
  - threads
  - devices
  - sms
  - warps
  - lanes
- Useful to get the picture of the current state of the application

State: Software Point of View

- **(cuda-gdb) info cuda kernels**

Kernel	Dev	Grid	SMs Mask	GridDim	BlockDim	Name	Args
* 0	0	1	0x000002	(1,1,1)	(1,1,1)	krnl0	data0=20
1	1	1	0x000001	(1,1,1)	(1,1,1)	krnl1	data1=12

- **(cuda-gdb) info cuda blocks** (v4.0)

BlockIdx	To BlockIdx	Count	State
* (0,0,0)	(97,0,0)	98	running
(102,0,0)	(111,0,0)	10	running

- **(cuda-gdb) info cuda threads**

Blockidx	ThreadIdx	BlockIdx	ThreadIdx	Cnt	Virtual PC	Filename	Line
* (0,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	1	0x05ae3168	foo.cu	383
(1,0,0)	(0,0,0)	(98,0,0)	(0,0,0)	98	0x05ae30a8	foo.cu	380
(102,0,0)	(0,0,0)	(111,0,0)	(0,0,0)	10	0x05ae30a8	foo.cu	380

State: Hardware Point of View

- **(cuda-gdb) info cuda devices**

Dev	Desc	SM Type	SMs	Warps/SM	Lanes/Warp	Regs/Lane	Active Mask
* 0	gf100	sm_20	14	48	32	64	0x00003fff
1	gt200	sm_13	30	32	32	128	0x00000000

- **(cuda-gdb) info cuda sms**

SM	Active Mask
* 0	0x000000000000003f

- **(cuda-gdb) info cuda warps**

Wp	Active Mask	Diverg Mask	Active PC	Kernel	BlockIdx
* 0	0xffffffe0	0x0000001f	0x0000638	1	(0,0,0)
1	0x00000000	0x00000000	n/a	n/a	n/a

- **(cuda-gdb) info cuda lanes**

Ln	State	Physical PC	ThreadIdx
0	divergent	0x000000c8	(0,0,0)

Accessing Variables Contents

- Use the standard print GDB command
  - **(cuda-gdb) print my_variable**
    - $1 = 3
- Variable must be live
  - compiler optimizes code, even with debug builds
  - required because of resource constraints
  - if variable not live at some location, try at another location
- Write a variable
  - **(cuda-gdb) print my_variable = 5**
    - $2 = 5

Accessing Memory Contents

- Use the standard print GDB command
  - **`(cuda-gdb) print *my_pointer`**
    - $1 = 3
- May require storage specifier when ambiguous
  - @generic
  - @global
  - @shared
  - @local
  - @texture
  - @parameter
- Textures
  - read-only
  - must be cast to the type of the array they are bound to
  - indexed like standard multi-dimensional C arrays

Accessing Memory Contents

- `(cuda-gdb) print my_local_variable`
  - `$1 = 3`
- `(cuda-gdb) print * (@global int *) my_pointer`
  - `$2 = 5`
- `(cuda-gdb) print ((@texture float **) my_texture)[0][3]`
  - `$3 = 2.5`

Accessing Hardware Registers

- CUDA Registers
  - virtual PC: $pc (read-only)
  - SASS registers: $R0, $R1, …
- Show all registers
  - **(cuda-gdb) info registers**
- Show a list of registers
  - **(cuda-gdb) info registers R2 R35**
- Modify one register
  - **(cuda-gdb) print $R3 = 3**

Tips

- Always check the return code of the CUDA API routines
- Use printf from the device code
  - make sure to synchronize so buffers are flushed
- To hide devices, launch the application with CUDA_VISIBLE_DEVICES = 0, 1
- To increase determinism, launch the kernels synchronously with the environment variable CUDA_LAUNCH_BLOCKING = 1