CUDA DEBUGGING TOOLS IN CUDA8

Vyas Venkataraman, Kudbudeen Jalaludeen, April 6, 2016
AGENDA

General debugging approaches
Cuda-gdb
Demo
CUDA API CHECKING

CUDA calls are asynchronous

Errors returned by any subsequent call

Check return status of CUDA API calls

CUDA Runtime API cudaError_t

CUDA Driver API Curesult

CUDA-GDB and CUDA-MEMCHECK have modes to do these checks
PRINTF()

Device side printf()
Output flushed to screen at explicit sync points
No inter-thread ordering guarantee
Increase backing storage `cudaDeviceSetLimit(cudaLimitPrintfFifoSize, size)`
Include stdio.h to use in program
ASSERT()

Stops device if conditional evaluates to 0

Error message printed to stderr

All subsequent CUDA API calls will return error

Include assert.h to use in program
NVCC COMPILER OPTIONS

Device side debug : -G
   Full debug information (variables, functions, line number)
   Disables optimization

Line number information only : -lineinfo
   No variable, function debug information
   No impact on optimizations
CUDA-MEMCHECK
Functional correctness checking tool suite

Part of CUDA toolkit

Multiple tools

- **memcheck**: reports out of bounds/misaligned memory access errors
- **racecheck**: identifies races on __shared__ memory
- **initcheck**: usage of uninitialized global memory
- **synccheck**: identify invalid usage of __syncthreads() in applications

CUDA-GDB
WHAT IS CUDA-GDB

Overview

Command line source and assembly (SASS) level debugger
Feature parity with Nsight Eclipse Edition
Simultaneous CPU and GPU debugging
Inspect and modify memory, register, variable state
Control program execution
Runtime GPU error detection
Support for multiple GPUs, multiple contexts, multiple kernels
LAUNCHED CUDA-GDB

Overview

Shipped as part of CUDA toolkit
Supports all CUDA capable GPUs
Supported on CUDA supported Linux distributions
Binary called cuda-gdb, accepts standard GDB command line parameters

```
$ cuda-gdb ./my_app
$ cuda-gdb --pid pid_to_attach_to
```
EXECUTION CONTROL

Usage

Identical to using GDB

Launching application

(cuda-gdb) run

Resume application after a breakpoint

(cuda-gdb) continue

Kill application

(cuda-gdb) Kill

Interrupt the application: CTRL + C
BREAKPOINTS

Usage

By name

(cuda-gdb) break bitreverse
(cuda-gdb) break _Z10bitreversePv

By file name and line number

(cuda-gdb) break bitreverse.cu:10

By address

(cuda-gdb) break *0xaf2468

At every CUDA kernel launch

(cuda-gdb) set cuda break_on_launch application
CONDITIONAL BREAKPOINTS

Usage

Breakpoint is reported if condition is met
Condition evaluated for all threads
Condition follows C/C++ syntax

(cuda-gdb) break bitreverse if (threadIdx.x==33)
(cuda-gdb) condition bpnum (threadIdx.x==33)
THREAD FOCUS
Overview

Needed for thread specific commands

7 dimensional logical value to identify a thread

Focus components

- **kernel**: Unique identifier, assigned to each kernel launch
- **block**: 3 dimensional block index `blockIdx.{x,y,z}`
- **thread**: 3 dimensional thread index `threadIdx.{x,y,z}`
THREAD FOCUS
Get current focus

Omitted focus components ignored

```
(cuda-gdb) cuda kernel block thread
kernel 2 block (2,0,0) thread (5,0,0)

(cuda-gdb) cuda thread
thread (5,0,0)
```
THREAD FOCUS
Listing GPU state

List all kernels, blocks, threads

(cuda-gdb) info cuda kernels
(cuda-gdb) info cuda blocks
(cuda-gdb) info cuda threads

Optional component specified focus filter

(cuda-gdb) info cuda threads kernel 0 block (1,0,0) thread(31,0,0)
(cuda-gdb) info cuda threads block (2,0,0)
(cuda-gdb) info cuda threads kernel 0
THREAD FOCUS

Switching focus

Specify target focus

Omitted components are assumed to stay same

(cudagdb) cuda kernel 1 block 1,0,0 thread 5,7,0

(cudagdb) cuda kernel 2 block 4

(cudagdb) cuda thread 8
VARIABLES AND MEMORY

Read a source variable or address

```
(cuda-gdb) print my_variable
(cuda-gdb) print *0x506b00000
```

Write a source variable or address

```
(cuda-gdb) set my_variable = 2
(cuda-gdb) set *0x506b00000 = 3.0
```

Access GPU memory segments using specifiers

```
@global, @shared, @local, @generic, @texture, @parameter, @managed
```
ATTACHING WITH CUDA-GDB

No special environment variables required

Full cuda-gdb functionality available after attach

```
(cuda-gdb) attach PID

$ cuda-gdb process_image PID
```

Detach and resume application execution
ATTACH ON GPU EXCEPTIONS

Allows CUDA application to wait when GPU exception is hit

Run application with environment variable CUDA_DEVICE_WAITS_ON_EXCEPTION=1

```
$ CUDA_DEVICE_WAITS_ON_EXCEPTION=1 ./my_app
```

On GPU exception, message is printed

Can attach with cuda-gdb
GENERATING GPU COREDUMP

GPU Coredump

Enabled via environment variable

```
$ CUDA_ENABLE_COREDUMP_ON_EXCEPTION=1 ./my_app
```

By default, GPU coredump causes a CPU coredump

Use environment variable `CUDA_ENABLE_CPU_COREDUMP_ON_EXCEPTION` to disable

Default filename `core_TIMESTAMP_HOSTNAME_PID.nvcudmp`

Environment variable `CUDA_COREDUMP_FILE` to set custom filename

Special format specifiers: `%p` (PID) ; `%h` (hostname) ; `%t` (timestamp)
LOADING A GPU COREDUMP

GPU Coredump

Use target cudacore

```
(cuda-gdb) target cudacore gpu_coredump_file.nvcudmp
```

Loading both CPU and GPU coredump files simultaneously

```
(cuda-gdb) target core core.cpu gpu_coredump_file.nvcudmp
```
CUDA ERROR REPORTING

**Usage**

CUDA API errors can be displayed, stopped on or hidden

```
(cu-gdb) set cuda api failures [ignore | stop | hide]
```

Enhanced interoperation with cuda-memcheck

```
(cu-gdb) set cuda memcheck on
```
CUDA SPECIAL OPTIONS

Read the CUDA Dynamic Parallelism (CDP) launch information

```
(cudagdb) info cuda launch trace
```

Read the list of allocations created via __managed__

```
(cudagdb) info cuda managed
```
WHAT’S NEW IN CUDA 8.0
NEW FEATURES IN CUDA 8.0

Support for Pascal SM 6.0
Support for Pascal Unified Memory
Compute Preemption Debugging
COMPUTE PREEMPTION

Functionality available on Pascal and newer GPUs
Periodic preemption of compute contexts
Enables applications with long running compute threads
Always enabled on Pascal GPUs
DEBUG ON DISPLAY GPU WITH NO RESTRICTIONS
RUN GRAPHICS APPLICATIONS WHILE DEBUGGING CUDA APPLICATIONS
RUN MULTIPLE DEBUGGER SESSIONS
DEBUG CUDA/OPENGL INTEROP APPLICATIONS
UNIFIED MEMORY

Information on hitting a process signal on managed memory
Support to read static __managed__ variables
For cudaMallocManaged() allocated memory, no extra migrations
COMPUTE PREEMPTION DEBUGGING

Set up

Single Machine(Ubuntu) with single Pascal GPU

Full support for all cuda-gdb features
ADDITIONAL RESOURCES

NVIDIA Nsight: http://www.nvidia.com/nsight


S6176 - Inside Pascal Talk

S6224 - CUDA8 and Beyond

S6810 - Optimizing Application Performance with CUDA Profiling Tools

Thu. 4/7, 10:00am, Room 211B

L6135A & L6135B - Jetson Developer Tools Lab

Wed. 4/6, 1:30pm & 3:30pm, Room 210C
THANK YOU

JOIN THE CONVERSATION
#GTC16  

JOIN THE NVIDIA DEVELOPER PROGRAM AT developer.nvidia.com/join