Debugging CUDA Applications on Linux and Mac

Vyas Venkataraman
Overview

- CUDA 101
  - Execution
  - Memory hierarchy
- Program Self Validation
- Debug Tools
  - CUDA-MEMCHECK
  - CUDA-GDB and Nvidia® Nsight™ Eclipse Edition
- Demo
CUDA 101: Threads, Blocks, Grids

- Threads are grouped into blocks
- Blocks are grouped into a grid
- A kernel is executed as a grid of blocks of threads
CUDA 101: Synchronization

1. First set of threads arrive
2. Second set of threads arrive
3. All threads resume

___syncthreads() enforces synchronization within a block
   Threads wait until all other threads in the same block have arrived
CUDA 101: Memory hierarchy

- **Thread:**
  - Registers
  - Local memory

- **Block of threads:**
  - Shared memory

- **All blocks:**
  - Global memory
A Skeletal CUDA Program

```c
__global__ void kernel(...) {
    // Do some device work
}

int main(...) {
    // Do some host activity
    cudaMalloc(&d_ptr, sz);
    cudaMemcpy(d_ptr, h_ptr, hostToDev);
    kernel<<<...>>>();
    cudaMemcpy(h_ptr, d_ptr, devToHost);
}
```
Debugging Applications

- **Effect**
  - Incorrect output
  - Hangs
  - Launch Failure
  - Crash

- **Isolation**
  - Prints
  - Assertions
  - Error checks
  - Tools

- **Cause**
  - Program logic
Program Self Validation

- Device side printf()
- Data validation
- CUDA API call check
- Device side assert()
Device side printf()

- SM 2.0 (Fermi) and above only
- C-style format string
  - Must match format string used on host
- Buffered output
  - Flushes only at explicit sync points
- Unordered
  - Think of multi threaded output
- Change the backing global memory storage
  - `cudaDeviceSetLimit(cudaLimitPrintFifoSize, size_t size);`
Device side printf() usage

- Include the `stdio.h` header
- Compile the app for Fermi:
  `nvcc -arch=compute_20 -o output test.cu`
- Run

```
$ ./demo_Printf
Var:42
```

```c
#include <stdio.h>

__device__ int var = 42;

__global__ void kernel(void)
{
    if (threadIdx.x == 0)
        printf("var:%d\n", var);
}

int main(void)
{
    kernel<<<1,1>>>();
    cudaDeviceSynchronize();
    cudaDeviceReset();
}
```
Data Validation

- Checking inputs when reading in
  - Validate values to be within expected ranges
  - Floating point NaN checks

- Output checks
  - Compute results on host vs device
  - Sanity check on output range
  - Floating point NaN/Inf checks
Data Validation Example

- Simple sanity check on input
- Expects all inputs to be normal or subnormal floats
- Depends on program

```c
bool check(float *arr, size_t elt) {
    for (int i = 0; i < elt; ++i)
        if (!isfinite(arr[i]))
            return false;
    return true;
}

int main(...) {
    float *a = malloc(elt * sizeof(*a));
    fread(a, sizeof(*a), elt, fil);
    if (!check(arr, elt))
        exit(1);
}
```
CUDA API Call

- Asynchronous calls
  - Errors returned by any subsequent call
  - Error state flushed once the device is synchronized
  - Program exit is not a synchronization point

- Check return status of API calls
  - CUDA Runtime API calls return cudaError_t
  - CUDA Driver API calls return CUresult

- CUDA-GDB and CUDA-MEMCHECK will perform these checks
CUDA API Call Checking

- Use macros
- Check all CUDA API calls
- Use cudaGetLastError to see the last error.

```c
#define CHECK(x) do {
    cudaError_t err = (x);
    if (err != cudaSuccess) {
        printf("API error failed \"%s:%d Returned:%d\n", __FILE__, __LINE__, err);
        exit(1);
    } while(0)

    int main(...) {
    ...
    CHECK(cudaMalloc(&d_ptr, sz));
    }
```
Device side assert()

- SM 2.0 (Fermi) and above only
- Stops if conditional == 0
- Prints the error message to stderr
- Printf()’s rules for flushing apply
- Stops all subsequent host side calls with cudaErrorAssert
Device side assert() usage

- Include the *assert.h* header
- Compile the app for Fermi:
  `nvcc -arch=compute_20 -o output test.cu`
- Run

```c
#include <assert.h>
__device__ int var;
__global__ void kernel(void)
{
    assert(threadId.x <= 16);
}

int main(void)
{
    kernel<<<1,18>>>();
    cudaDeviceSynchronize();
    cudaDeviceReset();
}
```

```bash
$ ./demo_assert
tmp/test_assert.cu:7: void kernel(): block: [0,0,0],
thread: [17,0,0] Assertion `threadIdx.x <=16` failed.
```

```c
./demo_assert /tmp/test_assert.cu:7: void kernel(): block: [0,0,0],
thread: [17,0,0] Assertion `threadIdx.x <=16` failed.
```
CUDA Debugging Solutions

CUDA-Memcheck
(Linux, Mac, & Windows)

NVIDIA® Nsight™
Eclipse Edition (Linux & Mac)
Visual Studio Edition (Windows)

CUDA-GDB
(Linux & Mac)

Allinea DDT

Rogue Wave TotalView
What is CUDA-MEMCHECK?

- “Why did my kernel fail?”
- Functional correctness tool suite
- Run time error checker: *memcheck*
  - Precise errors: Memory access
  - Imprecise errors: Hardware reported (SM 2.0+)
- Shared memory hazard checker: *racecheck*
- Cross platform: Linux, Mac, Windows
- Also integrated into cuda-gdb (Linux / Mac Only)
CUDA-MEMCHECK Features

- Misaligned and out of bounds memory access (*memcheck*)
- Hardware error reporting (*memcheck*)
- Shared memory hazard detection (*racecheck*)
- Device side malloc() / free() error checking
- Device heap allocation leak checking
- Device + Host stack back traces
- CUDA API error checking
- Name demangling (with parameters) for kernels
Running CUDA-MEMCHECK

- Standalone

```bash
$ cuda-memcheck [options] <my_app> <my_app_options>
```

- Default to `memcheck` tool

- Detects misaligned and out of bound access in GPU memory

```plaintext
Invalid __global__ read of size 4
  at 0x000000b8 in basic.cu:27:kernel2
  by thread (5,0,0) in block (3,0,0)
  Address 0x05500015 is misaligned
```
Running CUDA-MEMCHECK

- Imprecise errors

Out-of-range Shared or Local Address
at 0x00000798 in kernel1
by thread (0,0,0) in block (0,0,0)

- Device heap - double free() / invalid free() errors

- Leak checking
  - Not been free() / cudaFree()’d at context destroy

- Multiple precise errors using continue mode
__global__ int bcast(void) {
    int x;
    __shared__ int a;
    if (threadIdx.x == WRITER)
        a = threadIdx.x;
    x = a;
    // do some work
}

Thread 1

Thread 2

a
Sharing data between threads

- Data access hazard
- Data being read in thread 2 can be stale
- Need ordering
CUDA-MEMCHECK tool: Racecheck

- Built into cuda-memcheck
  - Use option --tool racecheck

  ```bash
  $ cuda-memcheck --tool racecheck <my_app> <my_app_options>
  ```

- Default: Byte accurate hazard reports
- Can provide source file and line
- Other useful options:
  - save to save output to a disk
  - print-level to control output
NVIDIA® NSIGHT™ ECLIPSE EDITION

- Debug View is powered by cuda-gdb
- Visualize device state
- Edit/Build/Debug/Profile
- Seamless CPU + GPU debugging
- Supported on Linux/Mac
CUDA-GDB Overview

- **What is it? What does it let you do?**
  - Command line source and assembly (SASS) level debugger
  - Feature set parity with Nsight Eclipse Edition
  - Simultaneous CPU and GPU debugging
    - Set Breakpoints and Conditional Breakpoints
    - Dump stack frames for thousands of CUDA threads
    - Inspect memory, registers, local/shared/global variables
  - Runtime Error Detection (stack overflow, …)
    - Can’t figure out why your kernel launch is failing? Run cuda-gdb!
    - Integrated cuda-memcheck support for increased precision
  - Supports multiple GPUs, multiple contexts, multiple kernels
CUDA-GDB Overview

- Which hardware does it support?
  - All CUDA-capable GPUs SM1.1 and beyond
  - Compatible with NVIDIA Optimus laptops

- Which platforms does it support?
  - All CUDA-supported Linux distributions
  - Mac OS X
  - 32-bit and 64-bit platforms
Execution Control

- Identical to host debugging:
  - Launch the application
    
    \[(\text{cuda-gdb}) \text{ run}\]
  - Resume the application (all host threads and device threads)
    
    \[(\text{cuda-gdb}) \text{ continue}\]
  - Kill the application
    
    \[(\text{cuda-gdb}) \text{ kill}\]
  - Interrupt the application: CTRL-C
Execution Control

- Single-Stepping
  - Applies to 32 threads at a time (a warp)

<table>
<thead>
<tr>
<th>Single-Stepping</th>
<th>At the source level</th>
<th>At the assembly level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Over function calls</td>
<td>next</td>
<td>nexti</td>
</tr>
<tr>
<td>Into function calls</td>
<td>step</td>
<td>stepi</td>
</tr>
</tbody>
</table>

- Behavior varies when stepping `__syncthreads()`

<table>
<thead>
<tr>
<th>PC at a barrier?</th>
<th>Single-stepping applies to</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>All threads in the current <strong>block</strong>.</td>
<td>Required to step over the barrier.</td>
</tr>
<tr>
<td>No</td>
<td><strong>Active threads</strong> in the current warp.</td>
<td></td>
</tr>
</tbody>
</table>
### Breakpoints

- **By name**
  - `(cuda-gdb) break my_kernel`
  - `(cuda-gdb) break _Z6kernelIfiEvPT_PT0`

- **By file name and line number**
  - `(cuda-gdb) break acos.cu:380`

- **By address**
  - `(cuda-gdb) break *0x3e840a8`
  - `(cuda-gdb) break *$pc`

- **At every kernel launch**
  - `(cuda-gdb) set cuda break_on_launch application`
Conditional Breakpoints

- Only reports hit breakpoint if condition is met
  - All breakpoints are still hit
  - Condition is evaluated every time for all the threads

- Condition
  - C/C++ syntax
  - supports built-in variables (blockIdx, threadIdx, ...)

(cuda-gdb) break acos.cu:380 if (...)(cuda-gdb) break acos.cu:380 if (...)
Thread Focus

- Some commands apply only to the thread in focus
  - Print local or shared variables
  - Print registers
  - Print stack contents

- Components
  - Kernel : unique, assigned at kernel launch time
  - Block : the application blockIdx
  - Thread : the application threadIdx
Thread Focus Nsight

CPU and GPU threads, call stacks
Thread Focus

- To switch focus to any currently running thread

(cuda-gdb) cuda kernel 2 block 1,0,0 thread 3,0,0
[Switching focus to CUDA kernel 2 block (1,0,0), thread (3,0,0)]

(cuda-gdb) cuda kernel 2 block 2 thread 4
[Switching focus to CUDA kernel 2 block (2,0,0), thread (4,0,0)]

(cuda-gdb) cuda thread 5
[Switching focus to CUDA kernel 2 block (2,0,0), thread (5,0,0)]

- Can also switch by HW coordinates : device/SM/warp/lane
Thread Focus

- To obtain the current focus:

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

(cuda-gdb) cuda thread
thread (5,0,0)
Threads

- To obtain the list of running threads for kernel 2:

  (cuda-gdb) info cuda threads kernel 2

<table>
<thead>
<tr>
<th>Block</th>
<th>Thread</th>
<th>To</th>
<th>Block</th>
<th>Thread</th>
<th>Cnt</th>
<th>PC</th>
<th>Filename</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(3,0,0)</td>
<td>(7,0,0)</td>
<td>32</td>
<td>0x7fae70</td>
<td>acos.cu</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td>(4,0,0)</td>
<td>(0,0,0)</td>
<td>(7,0,0)</td>
<td>(7,0,0)</td>
<td>32</td>
<td>0x7fae60</td>
<td>acos.cu</td>
<td>377</td>
</tr>
</tbody>
</table>

- Threads are displayed in (block,thread) ranges
- Divergent threads are in separate ranges
- The * indicates the range where the thread in focus resides
Stack Trace

- Applies to the thread in focus

```
(cuda-gdb) info stack
#0  fibo_aux (n=6) at fibo.cu:88
#1  0x7bbda0 in fibo_aux (n=7) at fibo.cu:90
#2  0x7bbda0 in fibo_aux (n=8) at fibo.cu:90
#3  0x7bbda0 in fibo_aux (n=9) at fibo.cu:90
#4  0x7bbda0 in fibo_aux (n=10) at fibo.cu:90
#5  0x7cfdb8 in fibo_main<<<(1,1,1),(1,1,1)>>>(...) at fibo.cu:95
```
Accessing Variables and Memory

Variables, breakpoints, CUDA kernel, thread, block, etc. state
Accessing Variables and Memory

- Read a source variable

(cuda-gdb) print my_variable
$1 = 3
(cuda-gdb) print &my_variable
$2 = (@global int *) 0x200200020

- Write a source variable

(cuda-gdb) print my_variable = 5
$3 = 5

- Access any GPU memory segment using storage specifiers
  - @global, @shared, @local, @generic, @texture, @parameter
Hardware Registers

- CUDA Registers
  - virtual PC: $pc (read-only)
  - SASS registers: $R0, $R1, ...

- Show a list of registers (blank for all)

```
(cuda-gdb) info registers R0 R1 R4
R0  0x6  6
R1  0xffffc68 16776296
R4  0x6  6
```

- Modify one register

```
(cuda-gdb) print $R3 = 3
```
Code Disassembly

(cuda-gdb) x/10i $pc
0x123830a8 <__Z9my_kernel+8>: MOV R0, c [0x0] [0x8]
0x123830b0 <__Z9my_kernel+16>: MOV R2, c [0x0] [0x14]
0x123830b8 <__Z9my_kernel+24>: IMUL.U32.U32 R0, R0, R2
0x123830c0 <__Z9my_kernel+32>: MOV R2, R0
0x123830c8 <__Z9my_kernel+40>: S2R R0, SR_CTAid_X
0x123830d0 <__Z9my_kernel+48>: MOV R0, R0
0x123830d8 <__Z9my_kernel+56>: MOV R3, c [0x0] [0x8]
0x123830e0 <__Z9my_kernel+64>: IMUL.U32.U32 R0, R0, R3
0x123830e8 <__Z9my_kernel+72>: MOV R0, R0
0x123830f0 <__Z9my_kernel+80>: MOV R0, R0
GPU Attach

CUDA-GDB/NSIGHT EE

CPU threads
GPU kernels, blocks, threads
CPU + GPU memory state
CPU + GPU register state

Attach at any point in time!
GPU Attach

- Run your program at full speed, then attach with cuda-gdb/Nsight EE
- No environment variables required!
- Inspect CPU and GPU state at any point in time
  - List all resident CUDA kernels
  - Utilize all existing CUDA-GDB commands
- Attach to CUDA programs forked by your application
- Detach and resume CPU and GPU execution
Attaching Nsight to a running CUDA process

1. Run your program, as usual

   ```
   $ myCudaApplication
   ```

2. Attach with Nsight EE, and see what’s going on
Attaching CUDA-GDB to a running CUDA process

1. Run your program, as usual

```
$ myCudaApplication
```

2. Attach with cuda-gdb, and see what’s going on

```
$ cuda-gdb myCudaApplication PID
```

```
Program received signal SIGTRAP, Trace/breakpoint trap.
[Switching focus to CUDA kernel 0, grid 2, block (0,0,0), thread (0,0,0),
device 0, sm 11, warp 1, lane 0]

0xae6688 in acos_main<<<(240,1,1),(128,1,1)>>> (parms=...) at acos.cu:383
383     while (!flag);
(cuda-gdb) p flag
$1 = 0
```
Attaching Nsight EE on GPU Exceptions

1. Run your program, asking the GPU to wait on exceptions

   $ CUDA_DEVICE_WAITS_ON_EXCEPTION=1 myCudaApplication

2. Upon hitting a fault, the following message is printed

   The application encountered a device error and CUDA_DEVICE_WAITS_ON_EXCEPTION is set. You can now attach a debugger to the application for inspection.

3. Attach with Nsight EE, and see which kernel faulted
Attaching CUDA-GDB on GPU Exceptions

1. Run your program, asking the GPU to wait on exceptions

   $ CUDA_DEVICE_WAITS_ON_EXCEPTION=1 myCudaApplication

2. Upon hitting a fault, the following message is printed

   The application encountered a device error and CUDA_DEVICE_WAITS_ON_EXCEPTION is set. You can now attach a debugger to the application for inspection.

3. Attach with cuda-gdb, and see which kernel faulted

   $ cuda-gdb myCudaApplication PID

   Program received signal CUDA_EXCEPTION_10, Device Illegal Address.

   (cuda-gdb) info cuda kernels
   Kernel  Dev  Grid  SMs  Mask  GridDim  BlockDim  Name  Args
   • 0 0 1 0x00000800  (1,1,1)  (1,1,1)  exception_kernel  data=...
CUDA Error Reporting in Nsight EE

- CUDA API error reporting
- Enhanced interoperation with cuda-memcheck

![Configuring CUDA GDB in Nsight EE](image-url)
CUDA Error Reporting in CUDA-GDB

- CUDA API error reporting (three modes)
  1. Trace all CUDA APIs that return an error code (default)
  2. Stop in the debugger when any CUDA API fails
  3. Hide all CUDA API errors (do not print them)

```
warning: CUDA API error detected: cudaMalloc returned (0xb)
```

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

- Enhanced interoperation with cuda-memcheck

```
(cuda-gdb) set cuda memcheck on
```

```
Memcheck detected an illegal access to address (@global)0x500200028
```
New in CUDA 5.5

- CUDA-GDB / Nsight™ Eclipse Edition
  - Single GPU debugging on SM 3.5 - BETA (See Demo Booth @ GTC)
  - Remote Debugging
  - CUDA Dynamic Parallelism Launch Trace
  - ARM debugging support

- CUDA-MEMCHECK
  - Racecheck Report Analysis Mode
  - Asynchronous Output
New in 5.5 : Single GPU Debugging

- BETA feature in CUDA-GDB and in Nsight EE
- SM 3.5 Only
- CDP debugging supported
- Demo in exhibit hall
New in 5.5: CDP Launch Trace in Nsight EE

- Tree view of current kernel’s ancestors
New in 5.5: CDP Launch Trace in CUDA-GDB

- Examine ancestors of CDP launched kernels

```
(cuda-gdb) info cuda launch trace
   Lvl Kernel Dev Grid Status GridDim BlockDim Invocation
* #0 5 0 -7 Active (1,1,1) (1,1,1) cdp_launchtrace(int)(depth = 3)
 #1 3 0 -6 Sleeping (1,1,1) (1,1,1) cdp_launchtrace(int)(depth = 2)
 #2 2 0 -5 Sleeping (1,1,1) (1,1,1) cdp_launchtrace(int)(depth = 1)
 #3 1 0  2 Sleeping (1,1,1) (1,1,1) cdp_launchtrace(int)(depth = 0)
```
New in 5.5 : Racecheck Analysis Mode

- Invoke with

```bash
$ cuda-memcheck --tool racecheck --racecheck-report analysis <my_app> <my_app_options>
```

- Analyzes thousands of hazard reports to produce simple user guidance

```
Race reported between Write access at 0x00000018 in raceGroupBasic.cu: 13: Basic(void)
and Write access at 0x00000018 in raceGroupBasic.cu:13:Basic(void)
```
Demo time
Thank You

- CUDA 5.5: [http://www.nvidia.com/getcuda](http://www.nvidia.com/getcuda)
- Lab workshop @GTC (S3519 - Tuesday 13:00 Room 230A)
- Second session @GTC (S3038 - Wednesday 10:00 Room 210C)
- Recordings from GTCs ([http://gputechconf.com](http://gputechconf.com))
- Demo booth @GTC
- Experts Table @GTC
- Email: cudatools@nvidia.com