CUDA-MEMCHECK

DU-05355-001_v04 | April 21, 2011

User Manual

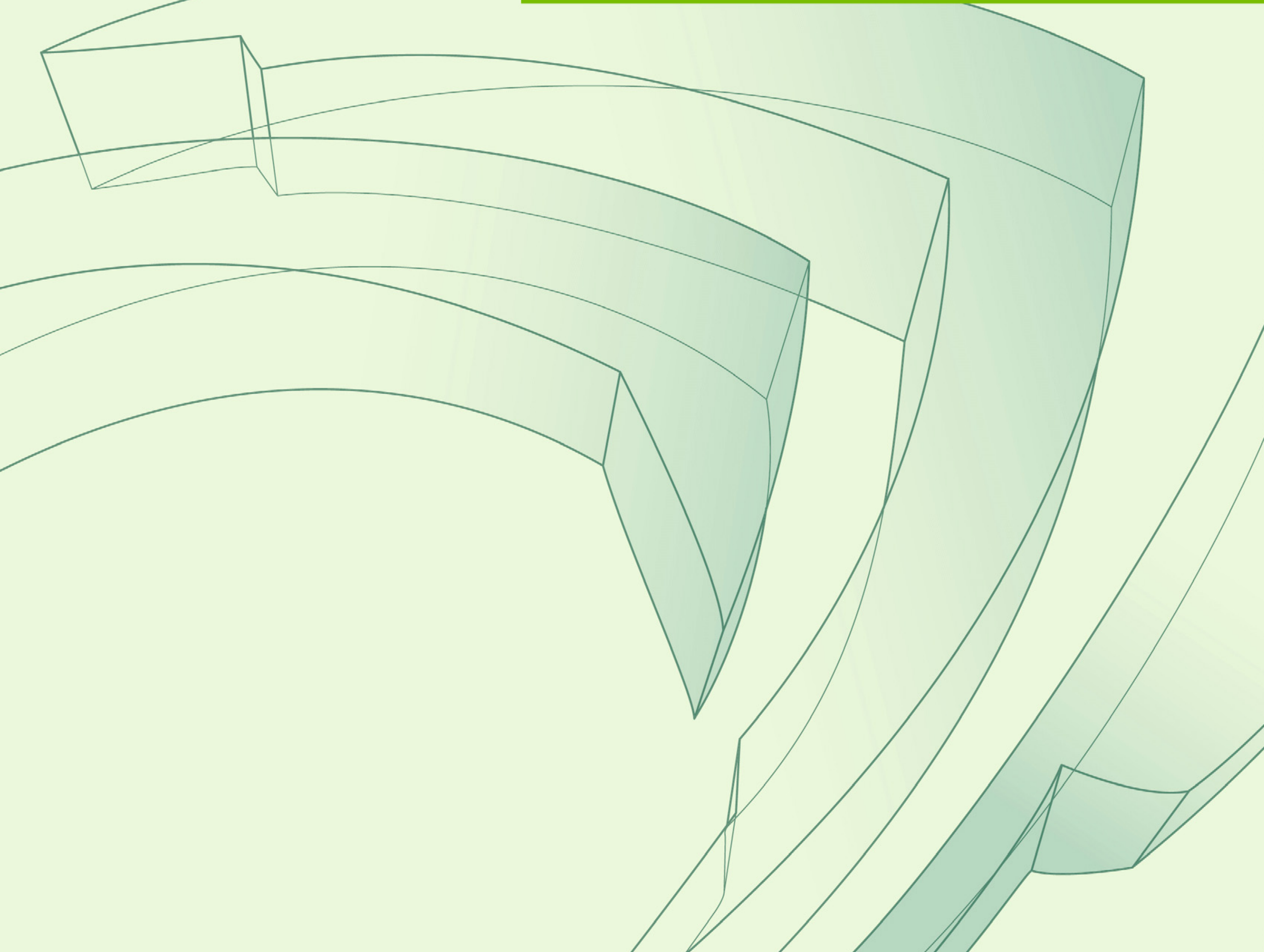

TABLE OF CONTENTS

01 INTRODUCTION

The CUDA debugger tool, `cuda-gdb`, includes a memory-checking feature for detecting and debugging memory errors in CUDA applications. This document describes that feature and tool, called `cuda-memcheck`.

About cuda-memcheck

Why cuda-memcheck

NVIDIA simplifies the debugging of CUDA programming errors with its powerful `cuda-gdb` hardware debugger. However, every programmer invariably encounters memory related errors that are hard to detect and time consuming to debug. The number of memory related errors increases substantially when dealing with thousands of threads. The `cuda-memcheck` tool is designed to detect such memory access errors in your CUDA application.

Supported error detection

The `cuda-memcheck` tool supports detection of out-of-bounds and misaligned global memory accesses.

For sm_20 and higher GPUs, `cuda-memcheck` also detects hardware exceptions. The supported exceptions are enumerated in Appendix A.

Note: Use of the continue flag is not supported after a hardware exception has been received.

Installation and cross-platform support

The standalone `cuda-memcheck` binary gets installed with `cuda-gdb` as part of the CUDA toolkit installation, and is supported on all CUDA supported platforms.

CUDA memory architecture

CUDA uses a segmented memory architecture that allows applications to access data in global, local, shared, constant, and texture memory.

A new unified addressing mode has been introduced in Fermi GPUs that allows data in global, local, and shared memory to be accessed with a generic 40-bit address.

02 USING CUDA-MEMCHECK

You can run cuda-memcheck as either a standalone tool or as part of cuda-gdb.

- "Using standalone cuda-memcheck" on page 3
- "Using integrated cuda-memcheck" on page 7

Using standalone cuda-memcheck

To run cuda-memcheck as a standalone tool, pass the application name as a parameter.

- Syntax:

  ```
  cuda-memcheck [options] [your-program] [your-program-options]
  ```

- Options field:
  - **-h** show this message
  - **--continue** try to continue running on memory access violations

    Refer to "Known Issues" on page 11 regarding use of the **-continue** flag.

You can execute either a debug or release build of your CUDA application with cuda-memcheck.

- Using a debug version of your application built with the **-g -G** option pair gives you additional information regarding the line number of the access violation.
- With a release version of the application, cuda-memcheck logs only the name of the kernel responsible for the access violation.

Sample Application Outputs

This section presents a walk-through of cuda-memcheck run with a simple application called **memcheck_demo**.

Note: Depending on the SM_type of your GPU, your system output may vary.

memcheck_demo.cu source code

```
#include <stdio.h>

__device__ int x;
__global__ void unaligned_kernel(void) {
    *(int*) ((char*)&x + 1) = 42;
}
__global__ void out_of_bounds_kernel(void) {
    *(int*) 0x87654320 = 42;
}

int main() {
    printf("Running unaligned_kernel\n");
    unaligned_kernel<<<1,1>>>();
    printf("Ran unaligned_kernel: %s\n",
            cudaGetErrorString(cudaGetLastError()));
    printf("Sync: %s\n", cudaGetErrorString(cudaThreadSynchronize()));

    printf("Running out_of_bounds_kernel\n");
    out_of_bounds_kernel<<<1,1>>>();
    printf("Ran out_of_bounds_kernel: %s\n",
            cudaGetErrorString(cudaGetLastError()));
    printf("Sync: %s\n", cudaGetErrorString(cudaThreadSynchronize()));

    return 0;
}
```

Application output without cuda-memcheck

When a CUDA application causes access violations, the kernel launch may terminate with an error code of unspecified launch failure or a subsequent cudathreadsynchronize call which will fail with an error code of unspecified launch failure.

This sample application is causing two failures but there is no way to detect where these kernels are causing the access violations, as illustrated in the following output:

```
$ ./memcheck_demo
      Running unaligned_kernel
      Ran unaligned_kernel: no error
      Sync: unspecified launch failure
      Running out_of_bounds_kernel
      Ran out_of_bounds_kernel: no error
      Sync: unspecified launch failure
```

(Debug Build) Application output with cuda-memcheck

Now run this application with cuda-memcheck and check the output. We will use the **--continue** option to let cuda-memcheck continue executing the rest of the kernel after its first access violation.

In the output below the first kernel does not see the unspecified launch failure error since that was the only access violation that kernel executes, and with the **–continue flag** set, cuda-memcheck will force it to continue. Depending on the application error checking, with the **--continue** flag set cuda-memcheck can detect more than one occurrence of the errors across kernels, but reports only the first error per kernel.

```
$ cuda-memcheck --continue ./memcheck_demo
    ========= CUDA-MEMCHECK
    Running unaligned_kernel
    Ran unaligned_kernel: no error
    Sync: no error
    Running out_of_bounds_kernel
    Ran out_of_bounds_kernel: no error
    Sync: unspecified launch failure
    ========= Invalid write of size 4
    =========   at 0x00000028 in memcheck_demo.cu:5:unaligned_kernel
    =========   by thread (0,0,0) in block (0,0)
    ========= Address 0x00002c01 is misaligned
    =========
    ========= Invalid write of size 4
    =========   at 0x00000048 in memcheck_demo.cu:8:out_of_bounds_kernel
    =========   by thread (0,0,0) in block (0,0)
    ========= Address 0x87654320 is out of bounds
    =========
    ========= ERROR SUMMARY: 2 errors
```

(Debug Build) Application output with cuda-memcheck, without --continue

Now run this application with cuda-memcheck but without using the **--continue** option.

Without the **–continue** option, the first kernel shows the unspecified launch failure and only the first error gets reported by cuda-memcheck. In this case, after the access violation in the first kernel the application allows the second kernel to execute and there is application output for both kernels. Even so, the cuda-memcheck error is logged only for the first kernel.

```
$ cuda-memcheck ./memcheck_demo
      ========= CUDA-MEMCHECK
      Running unaligned_kernel
      Ran unaligned_kernel: no error
      Sync: unspecified launch failure
      Running out_of_bounds_kernel
      Ran out_of_bounds_kernel: unspecified launch failure
      Sync: unspecified launch failure
      ========= Invalid write of size 4
      =========     at 0x00000028 in memcheck_demo.cu:5:unaligned_kernel
      =========     by thread (0,0,0) in block (0,0)
      ========= Address 0x00002c01 is misaligned
      =========
      ========= ERROR SUMMARY: 1 error
```

(Release Build) Application output with cuda-memcheck

In this case, since the application is built in release mode, the cuda-memcheck output contains only the kernel names from the application causing the access violation. Though the kernel name and error type are detected, there is no line number information on the failing kernel.

```
$ cuda-memcheck ./memcheck_demo
      Ran unaligned_kernel: no error
      Sync: unspecified launch failure
      Running out_of_bounds_kernel
      Ran out_of_bounds_kernel: unspecified launch failure
      Sync: unspecified launch failure
      ========= Invalid write of size 4
      =========     at 0x00000018 in unaligned_kernel
      =========     by thread (0,0,0) in block (0,0)
      ========= Address 0x00002c01 is misaligned
      =========
      ========= ERROR SUMMARY: 1 error
```

Using integrated cuda-memcheck

You can execute `cuda-memcheck` from within `cuda-gdb` by using the following variable before running the application:

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

Integrated cuda-memcheck example

This example shows how to enable `cuda-memcheck` from within `cuda-gdb` and detect errors within the debugger so you can access the line number information and check the state of the variables.

In this example the unaligned kernel has a misaligned memory access in block 1 lane 1, which gets trapped as an illegal lane address at line 5 from within `cuda-gdb`.

```
(cuda-gdb) r
Starting program: memcheck_demo
[Thread debugging using libthread_db enabled]
[New process 23653]
Running unaligned_kernel
[New Thread 140415864006416 (LWP 23653)]
[Launch of CUDA Kernel 0 on Device 0]

Program received signal CUDA_EXCEPTION_1, Lane Illegal Address.
[Switching to CUDA Kernel 0 (<<<(0,0),(0,0,0)>>>)]
0x0000000000992e68 in unaligned_kernel <<<(1,1),(1,1,1)>>> () at
memcheck_demo.cu:5
5               *(int*) ((char*)&x + 1) = 42;
(cuda-gdb) p &x
$1 = (@global int *) 0x42c00
(cuda-gdb) c
Continuing.

Program terminated with signal CUDA_EXCEPTION_1, Lane Illegal Address.
The program no longer exists.
(cuda-gdb)
```

APPENDIX A HARDWARE EXCEPTION REPORTING

The cuda-memcheck tool will report hardware exceptions when run as a standalone or as part of cuda-gdb. The table below enumerates the supported exceptions, their precision and scope, as well as a brief description of their cause. For more detailed information, see the documentation for cuda-gdb.

Table A.1 CUDA Exception Codes

Exception code	Precision of the Error	Scope of the Error	Description
CUDA_EXCEPTION_1 : "Lane Illegal Address"	Precise	Per lane/thread error	This occurs when a thread accesses an illegal(out of bounds) global address.
CUDA_EXCEPTION_2 : "Lane User Stack Overflow"	Precise	Per lane/thread error	This occurs when a thread exceeds its stack memory limit.
CUDA_EXCEPTION_3 : "Device Hardware Stack Overflow"	Not precise	Global error on the GPU	This occurs when the application triggers a global hardware stack overflow. The main cause of this error is large amounts of divergence in the presence of function calls.
CUDA_EXCEPTION_4 : "Warp Illegal Instruction"	Not precise	Warp error	This occurs when any thread within a warp has executed an illegal instruction.

Table A.1 CUDA Exception Codes (continued)

Exception code	Precision of the Error	Scope of the Error	Description
CUDA_EXCEPTION_5 : “Warp Out-of-range Address”	Not precise	Warp error	This occurs when any thread within a warp accesses an address that is outside the valid range of local or shared memory regions.
CUDA_EXCEPTION_6 : “Warp Misaligned Address”	Not precise	Warp error	This occurs when any thread within a warp accesses an address in the local or shared memory segments that is not correctly aligned.
CUDA_EXCEPTION_7 : “Warp Invalid Address Space”	Not precise	Warp error	This occurs when any thread within a warp executes an instruction that accesses a memory space not permitted for that instruction.
CUDA_EXCEPTION_8 : “Warp Invalid PC”	Not precise	Warp error	This occurs when any thread within a warp advances its PC beyond the 40-bit address space.
CUDA_EXCEPTION_9 : “Warp Hardware Stack Overflow”	Not precise	Warp error	This occurs when any thread in a warp triggers a hardware stack overflow. This should be a rare occurrence.
CUDA_EXCEPTION_10 : “Device Illegal Address”	Not precise	Global error	This occurs when a thread accesses an illegal(out of bounds) global address.
CUDA_EXCEPTION_11 : “Lane Misaligned Address”	Precise	Per lane/thread error	This occurs when a thread accesses a global address that is not correctly aligned.

Table A.1 CUDA Exception Codes (continued)

Exception code	Precision of the Error	Scope of the Error	Description
"Unknown Exception"	Not precise	Global Error	The precise cause of the exception is unknown. Potentially, this may be due to Device Hardware Stack overflows or a kernel generating an exception very close to its termination.

APPENDIX B KNOWN ISSUES

The following are known issues with the current release.

- Kernel launches larger than 16MB are not currently supported by `cuda-memcheck` and may return erroneous results.
- Applications run much slower under `cuda-memcheck`.
- `cuda-memcheck` imposes blocking launches which means only one kernel executes at a time.
- Without `cuda-memcheck`, when an application causes an access violation the kernel launch could fail with an error code of Unspecified Launch Failure.
- When using the "**`--continue`**" flag, `cuda-memcheck` tries to continue execution of the kernel and you may see more than one error getting detected.
- Accesses to device side memory allocations, created by calling `malloc()` inside a kernel, are not checked by `cuda-memcheck.`
- Use of the continue flag is not supported after a hardware exception has been received.
- CUDA_MEMCHECK does not support CUDA applications that directly read or write OpenGL-mapped push buffers inside the device code.

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA, the NVIDIA logo, NVIDIA nForce, GeForce, NVIDIA Quadro, NVDVD, NVIDIA Personal Cinema, NVIDIA Soundstorm, Vanta, TNT2, TNT, RIVA, RIVA TNT, VOODOO, VOODOO GRAPHICS, WAVEBAY, Accuview Antialiasing, Detonator, Digital Vibrance Control, ForceWare, NVRotate, NVSensor, NVSync, PowerMizer, Quincunx Antialiasing, Sceneshare, See What You've Been Missing, StreamThru, SuperStability, T-BUFFER, The Way It's Meant to be Played Logo, TwinBank, TwinView and the Video & Nth Superscript Design Logo are registered trademarks or trademarks of NVIDIA Corporation in the United States and/or other countries. Other company and product names may be trademarks or registered trademarks of the respective owners with which they are associated.

Copyright