cuda-binary-utilities

Release 13.0

NVIDIA Corporation

Oct 02, 2025

Contents

CUDA Binary Utilities

The application notes for cuobjdump, nvdisasm, cu++filt, and nvprune.

This document introduces `cuobjdump`, `nvdisasm`, `cu++filt` and `nvprune`, four CUDA binary tools for Linux (x86, ARM and P9), Windows, Mac OS and Android.

Chapter 1. What is a CUDA Binary?

A CUDA binary (also referred to as cubin) file is an ELF-formatted file which consists of CUDA executable code sections as well as other sections containing symbols, relocators, debug info, etc. By default, the CUDA compiler driver `nvcc` embeds cubin files into the host executable file. But they can also be generated separately by using the "`-cubin`" option of `nvcc`. cubin files are loaded at run time by the CUDA driver API.

Note: For more details on cubin files or the CUDA compilation trajectory, refer to NVIDIA CUDA Compiler Driver NVCC.

Chapter 2. Differences between `cuobjdump` and `nvdisasm`

CUDA provides two binary utilities for examining and disassembling cubin files and host executables: `cuobjdump` and `nvdisasm`. Basically, `cuobjdump` accepts both cubin files and host binaries while `nvdisasm` only accepts cubin files; but `nvdisasm` provides richer output options.

Here's a quick comparison of the two tools:

Table 1: Table 1. Comparison of `cuobjdump` and `nvdisasm`

	cuobjdump	**nvdisasm**
Disassemble cubin	Yes	Yes

Extract ptx and extract and disassemble cubin from the following input files:
▶ Host binaries
  ▶ Executables
  ▶ Object files
  ▶ Static libraries
▶ External fatbinary files
	Yes	No

Control flow analysis and output	No	Yes
Advanced display options	No	Yes

Chapter 3. Command Option Types and Notation

This section of the document provides common details about the command line options for the following tools:

- *cuobjdump*
- *nvdisasm*
- *nvprune*

Each command-line option has a long name and a short name, which are interchangeable with each other. These two variants are distinguished by the number of hyphens that must precede the option name, i.e. long names must be preceded by two hyphens and short names must be preceded by a single hyphen. For example, `-I` is the short name of `--include-path`. Long options are intended for use in build scripts, where size of the option is less important than descriptive value and short options are intended for interactive use.

The tools mentioned above recognize three types of command options: boolean options, single value options and list options.

Boolean options do not have an argument, they are either specified on a command line or not. Single value options must be specified at most once and list options may be repeated. Examples of each of these option types are, respectively:

```
Boolean option : nvdisasm --print-raw <file>
Single value   : nvdisasm --binary SM100 <file>
List options   : cuobjdump --function "foo,bar,foobar" <file>
```

Single value options and list options must have arguments, which must follow the name of the option by either one or more spaces or an equals character. When a one-character short name such as `-I`, `-l`, and `-L` is used, the value of the option may also immediately follow the option itself without being seperated by spaces or an equal character. The individual values of list options may be separated by commas in a single instance of the option or the option may be repeated, or any combination of these two cases.

Hence, for the two sample options mentioned above that may take values, the following notations are legal:

```
-o file
-o=file
-Idir1,dir2 -I=dir3 -I dir4,dir5
```

For options taking a single value, if specified multiple times, the rightmost value in the command line will be considered for that option. In the below example, `test.bin` binary will be disassembled assuming `SM120` as the architecture.

```
nvdisasm.exe -b SM100 -b SM120 test.bin
nvdisasm warning : incompatible redefinition for option 'binary', the last value of
↪this option was used
```

For options taking a list of values, if specified multiple times, the values get appended to the list. If there are duplicate values specified, they are ignored. In the below example, functions `foo` and `bar` are considered as valid values for option `--function` and the duplicate value `foo` is ignored.

```
cuobjdump --function "foo" --function "bar" --function "foo" -sass  test.cubin
```

Chapter 4. cuobjdump

cuobjdump extracts information from CUDA binary files (both standalone and those embedded in host binaries) and presents them in human readable format. The output of cuobjdump includes CUDA assembly code for each kernel, CUDA ELF section headers, string tables, relocators and other CUDA specific sections. It also extracts embedded ptx text from host binaries.

For a list of CUDA assembly instruction set of each GPU architecture, see *Instruction Set Reference*.

4.1. Usage

cuobjdump accepts a single input file each time it's run. The basic usage is as following:

```
cuobjdump [options] <file>
```

To disassemble a standalone cubin or cubins embedded in a host executable and show CUDA assembly of the kernels, use the following command:

```
cuobjdump -sass <input file>
```

To dump cuda elf sections in human readable format from a cubin file, use the following command:

```
cuobjdump -elf <cubin file>
```

To extract ptx text from a host binary, use the following command:

```
cuobjdump -ptx <host binary>
```

Here's a sample output of cuobjdump:

```
$ cuobjdump -ptx -sass add.o

Fatbin elf code:
================
arch = sm_100
code version = [1,8]
host = linux
compile_size = 64bit

     code for sm_100
     .target sm_100

             Function : _Z3addPfS_S_
```

(continues on next page)

(continued from previous page)

```
    .headerflags    @"EF_CUDA_SM100 EF_CUDA_VIRTUAL_SM(EF_CUDA_SM100)"
    /*0000*/                   LDC R1, c[0x0][0x37c] ;       /* 0x0000df00ff017b82 */
                                                             /* 0x000fe20000000800 */
    /*0010*/                   S2R R9, SR_TID.X ;            /* 0x0000000000097919 */
                                                             /* 0x000e2e0000002100 */
    /*0020*/                   S2UR UR6, SR_CTAID.X ;        /* 0x00000000000679c3 */
                                                             /* 0x000e220000002500 */
    /*0030*/                   LDCU.64 UR4, c[0x0][0x358] ;  /* 0x00006b00ff0477ac */
                                                             /* 0x000e6e0008000a00 */
    /*0040*/                   LDC R0, c[0x0][0x360] ;       /* 0x0000d800ff007b82 */
                                                             /* 0x000e300000000800 */
    /*0050*/                   LDC.64 R2, c[0x0][0x380] ;    /* 0x0000e000ff027b82 */
                                                             /* 0x000eb00000000a00 */
    /*0060*/                   LDC.64 R4, c[0x0][0x388] ;    /* 0x0000e200ff047b82 */
                                                             /* 0x000ee20000000a00 */
    /*0070*/                   IMAD R9, R0, UR6, R9 ;        /* 0x0000000600097c24 */
                                                             /* 0x001fce000f8e0209 */
    /*0080*/                   LDC.64 R6, c[0x0][0x390] ;    /* 0x0000e400ff067b82 */
                                                             /* 0x000e220000000a00 */
    /*0090*/                   IMAD.WIDE R2, R9, 0x4, R2 ;   /* 0x0000000409027825 */
                                                             /* 0x004fcc00078e0202 */
    /*00a0*/                   LDG.E R2, desc[UR4][R2.64] ;  /* 0x0000000402027981 */
                                                             /* 0x002ea2000c1e1900 */
    /*00b0*/                   IMAD.WIDE R4, R9, 0x4, R4 ;   /* 0x0000000409047825 */
                                                             /* 0x008fcc00078e0204 */
    /*00c0*/                   LDG.E R5, desc[UR4][R4.64] ;  /* 0x0000000404057981 */
                                                             /* 0x000ea2000c1e1900 */
    /*00d0*/                   IMAD.WIDE R6, R9, 0x4, R6 ;   /* 0x0000000409067825 */
                                                             /* 0x001fc800078e0206 */
    /*00e0*/                   FADD R9, R2, R5 ;             /* 0x0000000502097221 */
                                                             /* 0x004fca0000000000 */
    /*00f0*/                   STG.E desc[UR4][R6.64], R9 ;  /* 0x0000000906007986 */
                                                             /* 0x000fe2000c101904 */
    /*0100*/                   EXIT ;                        /* 0x000000000000794d */
                                                             /* 0x000fea0003800000 */
    /*0110*/                   BRA 0x110;                    /* 0xfffffffc00fc7947 */
                                                             /* 0x000fc0000383ffff */
    /*0120*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*0130*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*0140*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*0150*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*0160*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*0170*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*0180*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*0190*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*01a0*/                   NOP;                          /* 0x0000000000007918 */
                                                             /* 0x000fc00000000000 */
    /*01b0*/                   NOP;                          /* 0x0000000000007918 */
```

(continues on next page)

(continued from previous page)

```
                                                                /* 0x000fc00000000000 */
    /*01c0*/                  NOP;                              /* 0x0000000000007918 */
                                                                /* 0x000fc00000000000 */
    /*01d0*/                  NOP;                              /* 0x0000000000007918 */
                                                                /* 0x000fc00000000000 */
    /*01e0*/                  NOP;                              /* 0x0000000000007918 */
                                                                /* 0x000fc00000000000 */
    /*01f0*/                  NOP;                              /* 0x0000000000007918 */
                                                                /* 0x000fc00000000000 */
             ..........
```

```
Fatbin ptx code:
================
arch = sm_100
code version = [8,8]
host = linux
compile_size = 64bit
compressed
ptxasOptions =

//
//
//
//
//
//

.version 8.8
.target sm_100
.address_size 64

//

.visible .entry _Z3addPfS_S_(
.param .u64 .ptr .align 1 _Z3addPfS_S__param_0,
.param .u64 .ptr .align 1 _Z3addPfS_S__param_1,
.param .u64 .ptr .align 1 _Z3addPfS_S__param_2
)
{
.reg .b32 %r<5>;
.reg .f32 %f<4>;
.reg .b64 %rd<11>;

ld.param.u64 %rd1, [_Z3addPfS_S__param_0];
ld.param.u64 %rd2, [_Z3addPfS_S__param_1];
ld.param.u64 %rd3, [_Z3addPfS_S__param_2];
cvta.to.global.u64 %rd4, %rd3;
cvta.to.global.u64 %rd5, %rd2;
cvta.to.global.u64 %rd6, %rd1;
mov.u32 %r1, %tid.x;
mov.u32 %r2, %ctaid.x;
mov.u32 %r3, %ntid.x;
mad.lo.s32 %r4, %r2, %r3, %r1;
mul.wide.s32 %rd7, %r4, 4;
add.s64 %rd8, %rd6, %rd7;
ld.global.f32 %f1, [%rd8];
```

(continues on next page)

(continued from previous page)

```
add.s64 %rd9, %rd5, %rd7;
ld.global.f32 %f2, [%rd9];
add.f32 %f3, %f1, %f2;
add.s64 %rd10, %rd4, %rd7;
st.global.f32 [%rd10], %f3;
ret;
}
```

As shown in the output, the `a.out` host binary contains cubin and ptx code for sm_100.

To list cubin files in the host binary use `-lelf` option:

```
$ cuobjdump a.out -lelf
ELF file    1: add_new.sm_100.cubin
ELF file    2: add_new.sm_120.cubin
ELF file    3: add_old.sm_100.cubin
ELF file    4: add_old.sm_120.cubin
```

To extract all the cubins as files from the host binary use `-xelf all` option:

```
$ cuobjdump a.out -xelf all
Extracting ELF file    1: add_new.sm_100.cubin
Extracting ELF file    2: add_new.sm_120.cubin
Extracting ELF file    3: add_old.sm_100.cubin
Extracting ELF file    4: add_old.sm_120.cubin
```

To extract the cubin named `add_new.sm_100.cubin`:

```
$ cuobjdump a.out -xelf add_new.sm_100.cubin
Extracting ELF file    1: add_new.sm_100.cubin
```

To extract only the cubins containing `_old` in their names:

```
$ cuobjdump a.out -xelf _old
Extracting ELF file    1: add_old.sm_100.cubin
Extracting ELF file    2: add_old.sm_120.cubin
```

You can pass any substring to `-xelf` and `-xptx` options. Only the files having the substring in the name will be extracted from the input binary.

To dump common and per function resource usage information:

```
$ cuobjdump test.cubin -res-usage

Resource usage:
 Common:
  GLOBAL:56 CONSTANT[3]:28
 Function calculate:
  REG:24 STACK:8 SHARED:0 LOCAL:0 CONSTANT[0]:472 CONSTANT[2]:24 TEXTURE:0 SURFACE:0
↪SAMPLER:0
 Function mysurf_func:
  REG:38 STACK:8 SHARED:4 LOCAL:0 CONSTANT[0]:532 TEXTURE:8 SURFACE:7 SAMPLER:0
 Function mytexsampler_func:
  REG:42 STACK:0 SHARED:0 LOCAL:0 CONSTANT[0]:472 TEXTURE:4 SURFACE:0 SAMPLER:1
```

Note that value for REG, TEXTURE, SURFACE and SAMPLER denotes the count and for other resources it denotes no. of byte(s) used.

4.2. Command-line Options

Table 2 contains supported command-line options of `cuobjdump`, along with a description of what each option does. Each option has a long name and a short name, which can be used interchangeably.

Table 1: Table 2. cuobjdump Command-line Options

Option (long)	Option (short)	Description
--all-fatbin	-all	Dump all fatbin sections. By default will only dump contents of executable fatbin (if exists), else relocatable fatbin if no executable fatbin.
--dump-elf	-elf	Dump ELF Object sections.
--dump-elf-symbols	-symbols	Dump ELF symbol names.
--dump-ptx	-ptx	Dump PTX for all listed device functions.
--dump-sass	-sass	Dump CUDA assembly for a single cubin file or all cubin files embedded in the binary.
--dump-resource-usage	-res-usage	Dump resource usage for each ELF. Useful in getting all the resource usage information at one place.
--extract-elf <partial file name>,...	-xelf	Extract ELF file(s) name containing <partial file name> and save as file(s). Use all to extract all files. To get the list of ELF files use -lelf option. Works with host executable/object/library and external fatbin. All dump and list options are ignored with this option.
--extract-ptx <partial file name>,...	-xptx	Extract PTX file(s) name containing <partial file name> and save as file(s). Use all to extract all files. To get the list of PTX files use -lptx option. Works with host executable/object/library and external fatbin. All dump and list options are ignored with this option.
--extract-text <partial file name>,...	-xtext	Extract text binary encoding file(s) name containing <partial file name> and save as file(s). Use 'all' to extract all files. To get the list of text binary encoding use -ltext option. All 'dump' and 'list' options are ignored with this option.
--function <function name>,...	-fun	Specify names of device functions whose fat binary structures must be dumped.
--function-index <function index>,...	-findex	Specify symbol table index of the function whose fat binary structures must be dumped.
--gpu-architecture <gpu architecture name>	-arch	Specify GPU Architecture for which information should be dumped. Allowed values for this option: sm_75, sm_80, sm_86, sm_87, sm_88, sm_89, sm_90, sm_90a, sm_100, sm_100a, sm_100f, sm_103, sm_103a, sm_103f, sm_110, sm_110a, sm_110f, sm_120, sm_120a, sm_120f, sm_121, sm_121a, sm_121f.
--help	-h	Print this help information on this tool.
--list-elf	-lelf	List all the ELF files available in the fatbin. Works with host executable/object/library and external fatbin. All other options are ignored with this flag. This can be used to select particular ELF with -xelf option later.
--list-ptx	-lptx	List all the PTX files available in the fatbin. Works with host executable/object/library and external fatbin. All other options are ignored with this flag. This can be used to select particular PTX with -xptx option later.
--list-text	-ltext	List all the text binary function names available in the fatbin. All other options are ignored with the flag. This

Chapter 5. nvdisasm

nvdisasm extracts information from standalone cubin files and presents them in human readable format. The output of nvdisasm includes CUDA assembly code for each kernel, listing of ELF data sections and other CUDA specific sections. Output style and options are controlled through nvdisasm command-line options. nvdisasm also does control flow analysis to annotate jump/branch targets and makes the output easier to read.

Note: nvdisasm requires complete relocation information to do control flow analysis. If this information is missing from the CUDA binary, either use the nvdisasm option -ndf to turn off control flow analysis, or use the ptxas and nvlink option -preserve-relocs to re-generate the cubin file.

For a list of CUDA assembly instruction set of each GPU architecture, see *Instruction Set Reference*.

5.1. Usage

nvdisasm accepts a single input file each time it's run. The basic usage is as following:

```
nvdisasm [options] <input cubin file>
```

Here's a sample output of nvdisasm:

```
    .elftype        @"ET_EXEC"

//--------------------- .nv.info                  --------------------------
    .section        .nv.info,"",@"SHT_CUDA_INFO"
    .align  4

......

//--------------------- .text._Z9acos_main10acosParams --------------------------
    .section    .text._Z9acos_main10acosParams,"ax",@progbits
    .sectioninfo    @"SHI_REGISTERS=14"
    .align    128
        .global     _Z9acos_main10acosParams
        .type       _Z9acos_main10acosParams,@function
        .size       _Z9acos_main10acosParams,(.L_21 - _Z9acos_main10acosParams)
        .other      _Z9acos_main10acosParams,@"STO_CUDA_ENTRY STV_DEFAULT"
_Z9acos_main10acosParams:
.text._Z9acos_main10acosParams:
        /*0000*/               MOV R1, c[0x0][0x28] ;
```

(continues on next page)

(continued from previous page)

```
        /*0010*/                   NOP;
        /*0020*/                   S2R R0, SR_CTAID.X ;
        /*0030*/                   S2R R3, SR_TID.X ;
        /*0040*/                   IMAD R0, R0, c[0x0][0x0], R3 ;
        /*0050*/                   ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;
        /*0060*/               @P0 EXIT ;
.L_1:
        /*0070*/                   MOV R11, 0x4 ;
        /*0080*/                   IMAD.WIDE R2, R0, R11, c[0x0][0x160] ;
        /*0090*/                   LDG.E.SYS R2, [R2] ;
        /*00a0*/                   MOV R7, 0x3d53f941 ;
        /*00b0*/                   FADD.FTZ R4, |R2|.reuse, -RZ ;
        /*00c0*/                   FSETP.GT.FTZ.AND P0, PT, |R2|.reuse, 0.5699, PT ;
        /*00d0*/                   FSETP.GEU.FTZ.AND P1, PT, R2, RZ, PT ;
        /*00e0*/                   FADD.FTZ R5, -R4, 1 ;
        /*00f0*/                   IMAD.WIDE R2, R0, R11, c[0x0][0x168] ;
        /*0100*/                   FMUL.FTZ R5, R5, 0.5 ;
        /*0110*/               @P0 MUFU.SQRT R4, R5 ;
        /*0120*/                   MOV R5, c[0x0][0x0] ;
        /*0130*/                   IMAD R0, R5, c[0x0][0xc], R0 ;
        /*0140*/                   FMUL.FTZ R6, R4, R4 ;
        /*0150*/                   FFMA.FTZ R7, R6, R7, 0.018166976049542427063 ;
        /*0160*/                   FFMA.FTZ R7, R6, R7, 0.046756859868764877319 ;
        /*0170*/                   FFMA.FTZ R7, R6, R7, 0.074846573173999786377 ;
        /*0180*/                   FFMA.FTZ R7, R6, R7, 0.16667014360427856445 ;
        /*0190*/                   FMUL.FTZ R7, R6, R7 ;
        /*01a0*/                   FFMA.FTZ R7, R4, R7, R4 ;
        /*01b0*/                   FADD.FTZ R9, R7, R7 ;
        /*01c0*/              @!P0 FADD.FTZ R9, -R7, 1.5707963705062866211 ;
        /*01d0*/                   ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;
        /*01e0*/              @!P1 FADD.FTZ R9, -R9, 3.1415927410125732422 ;
        /*01f0*/                   STG.E.SYS [R2], R9 ;
        /*0200*/              @!P0 BRA `(.L_1) ;
        /*0210*/                   EXIT ;
.L_2:
        /*0220*/                   BRA `(.L_2);
.L_21:
```

To get the control flow graph of a kernel, use the following:

```
nvdisasm -cfg <input cubin file>
```

`nvdisasm` is capable of generating control flow of CUDA assembly in the format of DOT graph description language. The output of the control flow from nvdisasm can be directly imported to a DOT graph visualization tool such as Graphviz.

Here's how you can generate a PNG image (`cfg.png`) of the control flow of the above cubin (`a.cubin`) with `nvdisasm` and Graphviz:

```
nvdisasm -cfg a.cubin | dot -ocfg.png -Tpng
```

Here's the generated graph:

To generate a PNG image (`bbcfg.png`) of the basic block control flow of the above cubin (`a.cubin`) with `nvdisasm` and Graphviz:

```
nvdisasm -bbcfg a.cubin | dot -obbcfg.png -Tpng
```

Fig. 1: Control Flow Graph

Here’s the generated graph:

nvdisasm is capable of showing the register (general and predicate) liveness range information. For each line of CUDA assembly, nvdisasm displays whether a given device register was assigned, accessed, live or re-assigned. It also shows the total number of registers used. This is useful if the user is interested in the life range of any particular register, or register usage in general.

Here’s a sample output (output is pruned for brevity):

```
                                                      // +-----------------+------+
                                                      // |      GPR        | PRED |
                                                      // |                 |      |
                                                      // |                 |      |
                                                      // |    000000000011 |      |
                                                      // |  # 012345678901 | # 01 |
                                                      // +-----------------+------+
    .global acos                                      // |                 |      |
    .type   acos,@function                            // |                 |      |
    .size   acos,(.L_21 - acos)                       // |                 |      |
    .other  acos,@"STO_CUDA_ENTRY STV_DEFAULT"        // |                 |      |
acos:                                                 // |                 |      |
.text.acos:                                           // |                 |      |
    MOV R1, c[0x0][0x28] ;                            // |  1  ^           |      |
    NOP;                                              // |  1  ^           |      |
    S2R R0, SR_CTAID.X ;                              // |  2 ^:           |      |
    S2R R3, SR_TID.X ;                                // |  3 :: ^         |      |
    IMAD R0, R0, c[0x0][0x0], R3 ;                    // |  3 x: v         |      |
    ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;      // |  2 v:           | 1 ^  |
@P0 EXIT ;                                            // |  2 ::           | 1 v  |
.L_1:                                                 // |  2 ::           |      |
     MOV R11, 0x4 ;                                   // |  3 ::         ^ |      |
     IMAD.WIDE R2, R0, R11, c[0x0][0x160] ;           // |  5 v:^^       v |      |
     LDG.E.SYS R2, [R2] ;                             // |  4 ::^        : |      |
     MOV R7, 0x3d53f941 ;                             // |  5 :::    ^   : |      |
     FADD.FTZ R4, |R2|.reuse, -RZ ;                   // |  6 ::v ^  :   : |      |
     FSETP.GT.FTZ.AND P0, PT, |R2|.reuse, 0.5699, PT; // |  6 ::v :  :   : | 1 ^  |
     FSETP.GEU.FTZ.AND P1, PT, R2, RZ, PT ;           // |  6 ::v :  :   : | 2 :^ |
     FADD.FTZ R5, -R4, 1 ;                            // |  6 ::  v^ :   : | 2 :: |
     IMAD.WIDE R2, R0, R11, c[0x0][0x168] ;           // |  8 v:^^:: :   v | 2 :: |
     FMUL.FTZ R5, R5, 0.5 ;                           // |  5 ::  :x :     | 2 :: |
 @P0 MUFU.SQRT R4, R5 ;                               // |  5 ::  ^v :     | 2 v: |
     MOV R5, c[0x0][0x0] ;                            // |  5 ::  :^ :     | 2 :: |
     IMAD R0, R5, c[0x0][0xc], R0 ;                   // |  5 x:  :v :     | 2 :: |
     FMUL.FTZ R6, R4, R4 ;                            // |  5 ::  v ^:     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.018166976049542427063 ;   // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.046756859868764877319 ;   // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.074846573173999786377 ;   // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R6, R7, 0.16667014360427856445 ;    // |  5 ::  : vx     | 2 :: |
     FMUL.FTZ R7, R6, R7 ;                            // |  5 ::  : vx     | 2 :: |
     FFMA.FTZ R7, R4, R7, R4 ;                        // |  4 ::  v  x     | 2 :: |
     FADD.FTZ R9, R7, R7 ;                            // |  4 ::     v ^   | 2 :: |
@!P0 FADD.FTZ R9, -R7, 1.5707963705062866211 ;        // |  4 ::     v ^   | 2 v: |
     ISETP.GE.AND P0, PT, R0, c[0x0][0x170], PT ;     // |  3 v:       :   | 2 ^: |
@!P1 FADD.FTZ R9, -R9, 3.1415927410125732422 ;        // |  3 ::       x   | 2 :v |
     STG.E.SYS [R2], R9 ;                             // |  3 ::       v   | 1 :  |
@!P0 BRA `(.L_1) ;                                    // |  2 ::           | 1 v  |
     EXIT ;                                           // |  1  :           |      |
.L_2:                                                 // +.................+......+
```

(continues on next page)

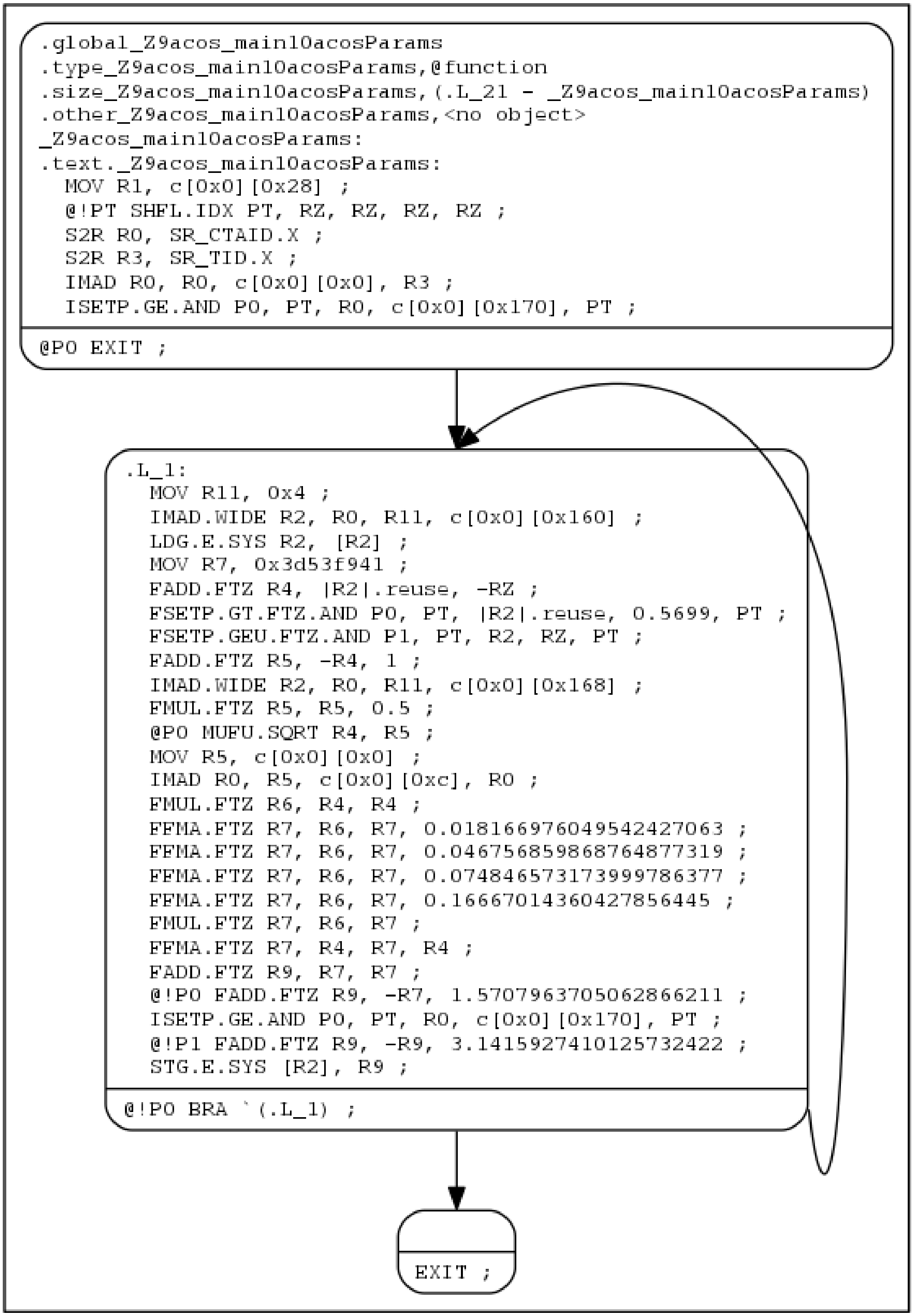

Fig. 2: Basic Block Control Flow Graph

(continued from previous page)

```
     BRA `(.L_2);                                          // |                  |      |
.L_21:                                                     // +------------------+------+
                                                           // Legend:
                                                           //     ^       : Register
↪assignment
                                                           //     v       : Register usage
                                                           //     x       : Register usage
↪and reassignment
                                                           //     :       : Register in use
                                                           //     <space> : Register not
↪in use
                                                           //     #       : Number of
↪occupied registers
```

`nvdisasm` is capable of showing line number information of the CUDA source file which can be useful for debugging.

To get the line-info of a kernel, use the following:

```
nvdisasm -g <input cubin file>
```

Here's a sample output of a kernel using `nvdisasm -g` command:

```
//--------------------- .text._Z6kernali          --------------------------
        .section        .text._Z6kernali,"ax",@progbits
        .sectioninfo    @"SHI_REGISTERS=24"
        .align  128
        .global         _Z6kernali
        .type           _Z6kernali,@function
        .size           _Z6kernali,(.L_4 - _Z6kernali)
        .other          _Z6kernali,@"STO_CUDA_ENTRY STV_DEFAULT"
_Z6kernali:
.text._Z6kernali:
        /*0000*/                   MOV R1, c[0x0][0x28] ;
        /*0010*/                   NOP;
    //## File "/home/user/cuda/sample/sample.cu", line 25
        /*0020*/                   MOV R0, 0x160 ;
        /*0030*/                   LDC R0, c[0x0][R0] ;
        /*0040*/                   MOV R0, R0 ;
        /*0050*/                   MOV R2, R0 ;
    //## File "/home/user/cuda/sample/sample.cu", line 26
        /*0060*/                   MOV R4, R2 ;
        /*0070*/                   MOV R20, 32@lo((_Z6kernali + .L_1@srel)) ;
        /*0080*/                   MOV R21, 32@hi((_Z6kernali + .L_1@srel)) ;
        /*0090*/                   CALL.ABS.NOINC `(_Z3fooi) ;
.L_1:
        /*00a0*/                   MOV R0, R4 ;
        /*00b0*/                   MOV R4, R2 ;
        /*00c0*/                   MOV R2, R0 ;
        /*00d0*/                   MOV R20, 32@lo((_Z6kernali + .L_2@srel)) ;
        /*00e0*/                   MOV R21, 32@hi((_Z6kernali + .L_2@srel)) ;
        /*00f0*/                   CALL.ABS.NOINC `(_Z3bari) ;
.L_2:
        /*0100*/                   MOV R4, R4 ;
        /*0110*/                   IADD3 R4, R2, R4, RZ ;
        /*0120*/                   MOV R2, 32@lo(arr) ;
        /*0130*/                   MOV R3, 32@hi(arr) ;
```

(continues on next page)

(continued from previous page)

```
        /*0140*/                   MOV R2, R2 ;
        /*0150*/                   MOV R3, R3 ;
        /*0160*/                   ST.E.SYS [R2], R4 ;
    //## File "/home/user/cuda/sample/sample.cu", line 27
        /*0170*/                   ERRBAR ;
        /*0180*/                   EXIT ;
.L_3:
        /*0190*/                   BRA `(.L_3);
.L_4:
```

nvdisasm is capable of showing line number information with additional function inlining info (if any). In absence of any function inlining the output is same as the one with `nvdisasm -g` command.

Here's a sample output of a kernel using `nvdisasm -gi` command:

```
//---------------------- .text._Z6kernali          --------------------------
    .section    .text._Z6kernali,"ax",@progbits
    .sectioninfo    @"SHI_REGISTERS=16"
    .align    128
        .global          _Z6kernali
        .type            _Z6kernali,@function
        .size            _Z6kernali,(.L_18 - _Z6kernali)
        .other           _Z6kernali,@"STO_CUDA_ENTRY STV_DEFAULT"
_Z6kernali:
.text._Z6kernali:
        /*0000*/                   IMAD.MOV.U32 R1, RZ, RZ, c[0x0][0x28] ;
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*0010*/                   UMOV UR4, 32@lo(arr) ;
        /*0020*/                   UMOV UR5, 32@hi(arr) ;
        /*0030*/                   IMAD.U32 R2, RZ, RZ, UR4 ;
        /*0040*/                   MOV R3, UR5 ;
        /*0050*/                   ULDC.64 UR4, c[0x0][0x118] ;
    //## File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*0060*/                   LDG.E R4, [R2.64] ;
        /*0070*/                   LDG.E R5, [R2.64+0x4] ;
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*0080*/                   LDG.E R0, [R2.64+0x8] ;
    //## File "/home/user/cuda/inline.cu", line 23
        /*0090*/                   UMOV UR6, 32@lo(ans) ;
        /*00a0*/                   UMOV UR7, 32@hi(ans) ;
    //## File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*00b0*/                   IADD3 R7, R4, c[0x0][0x160], RZ ;
    //## File "/home/user/cuda/inline.cu", line 23
        /*00c0*/                   IMAD.U32 R4, RZ, RZ, UR6 ;
    //## File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
```

(continues on next page)

(continued from previous page)

```
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*00d0*/                   IADD3 R9, R5, c[0x0][0x160], RZ ;
    //## File "/home/user/cuda/inline.cu", line 23
        /*00e0*/                   MOV R5, UR7 ;
    //## File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*00f0*/                   IADD3 R11, R0.reuse, c[0x0][0x160], RZ ;
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*0100*/                   IMAD.IADD R13, R0, 0x1, R7 ;
    //## File "/home/user/cuda/inline.cu", line 10 inlined at "/home/user/cuda/inline.
↪cu", line 17
    //## File "/home/user/cuda/inline.cu", line 17 inlined at "/home/user/cuda/inline.
↪cu", line 23
    //## File "/home/user/cuda/inline.cu", line 23
        /*0110*/                   STG.E [R2.64+0x4], R9 ;
        /*0120*/                   STG.E [R2.64], R7 ;
        /*0130*/                   STG.E [R2.64+0x8], R11 ;
    //## File "/home/user/cuda/inline.cu", line 23
        /*0140*/                   STG.E [R4.64], R13 ;
    //## File "/home/user/cuda/inline.cu", line 24
        /*0150*/                   EXIT ;
.L_3:
        /*0160*/                   BRA (.L_3);
.L_18:
```

nvdisasm can generate disassembly in JSON format.

For details on the JSON format, see *Appendix*.

To get disassembly in JSON format, use the following:

```
nvdisasm -json <input cubin file>
```

The output from nvdisasm -json will be in minified format. The sample below is after beautifying it:

```
[
    {
        "ELF": {
            "layout-id": 4,
            "ei_osabi": 51,
            "ei_abiversion": 7
        },
        "SM": {
            "version": {
                "major": 9,
                "minor": 0
            }
        },
        "SchemaVersion": {
            "major": 12,
```

(continues on next page)

(continued from previous page)

```
            "minor": 8,
            "revision": 0
        },
        "Producer": "nvdisasm V12.8.14 Build r570_00.r12.8/compiler.35033008_0",
        "Description": ""
    },
    [
        {
            "function-name": "foo",
            "start": 0,
            "length": 96,
            "other-attributes": [],
            "sass-instructions": [
                {
                    "opcode": "LDC",
                    "operands": "R1,c[0x0][0x28]"
                },
                {
                    "opcode": "MOV",
                    "operands": "R6,0x60"
                },
                {
                    "opcode": "ISETP.NE.U32.AND",
                    "operands": "P0,PT,R1,0x1,PT"
                },
                {
                    "opcode": "CALL.REL.NOINC",
                    "operands": "`(bar)",
                    "other-attributes": {
                        "control-flow": "True"
                    }
                },
                {
                    "opcode": "MOV",
                    "operands": "R8,R7"
                },
                {
                    "opcode": "EXIT",
                    "other-attributes": {
                        "control-flow": "True"
                    }
                }
            ]
        },
        {
            "function-name": "bar",
            "start": 96,
            "length": 32,
            "other-attributes": [],
            "sass-instructions": [
                {
                    "opcode": "STS.128",
                    "operands": "[UR5+0x400],RZ"
                },
                {
                    "opcode": "RET.REL.NODEC",
```

(continues on next page)

(continued from previous page)

```
                    "operands": "R18,`(foo)",
                    "other-attributes": {
                        "control-flow": "True"
                    }
                }
            ]
        }
    ]
]
```

5.2. Command-line Options

Table 3 contains the supported command-line options of `nvdisasm`, along with a description of what each option does. Each option has a long name and a short name, which can be used interchangeably.

Table 1: Table 3. nvdisasm Command-line Options

Option (long)	Option (short)	Description
--base-address <value>	-base	Specify the logical base address of the image to disassemble. This option is only valid when disassembling a raw instruction binary (see option --binary), and is ignored when disassembling an Elf file. Default value: 0.
--binary <SMxy>	-b	When this option is specified, the input file is assumed to contain a raw instruction binary, that is, a sequence of binary instruction encodings as they occur in instruction memory. The value of this option must be the asserted architecture of the raw binary. Allowed values for this option: SM75, SM80, SM86, SM87, SM88, SM89, SM90, SM90a, SM100, SM100a, SM103, SM103a, SM110, SM110a, SM120, SM120a, SM121, SM121a.
--cuda-function-index <symbol index>,...	-fun	Restrict the output to the CUDA functions represented by symbols with the given indices. The CUDA function for a given symbol is the enclosing section. This only restricts executable sections; all other sections will still be printed.
--emit-json	-json	Print disassembly in JSON format. This can be used along with the options --binary <SMxy> and --cuda-function-index <symbol index>,.... For details on the JSON format, see *Appendix*. However this is not compatible with options --print-life-ranges, --life-range-mode, --output-control-flow-graph and --output-control-flow-graph-with-basic-blocks.
--help	-h	Print this help information on this tool.
--life-range-mode	-lrm	This option implies option --print-life-ranges, and determines how register live range info should be printed. count: Not at all, leaving only the # column (number of live registers); wide: Columns spaced out for readability (default); narrow: A one-character column for each register, economizing on table width Allowed values for this option: count, narrow, wide.
--no-dataflow	-ndf	Disable dataflow analyzer after disassembly. Dataflow analysis is normally enabled to perform branch stack analysis and annotate all instructions that jump via the GPU branch stack with inferred branch target labels. However, it may occasionally fail when certain restrictions on the input nvelf/cubin are not met.
--no-vliw	-novliw	Conventional mode; disassemble paired instructions in normal syntax, instead of VLIW syntax.
--options-file <file>,...	-optf	Include command line options from specified file.
--output-control-flow-graph	-cfg	When specified output the control flow graph, where each node is a hyperblock, in a format consumable by graphviz tools (such as dot).
--output-control-flow-graph-with-basic-blocks	-bbcfg	When specified output the control flow graph, where each node is a basicblock, in a format consumable by graphviz tools (such as dot).
--print-code	-c	Only print code sections.

Chapter 6. Instruction Set Reference

This section contains instruction set reference for NVIDIA® GPU architectures.

6.1. Turing Instruction Set

The Turing architecture (Compute Capability 7.5) have the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- RX for registers
- URX for uniform registers
- SRX for special system-controlled registers
- PX for predicate registers
- c[X][Y] for constant memory

Table 6 lists valid instructions for the Turing GPUs.

Table 1: Table 6. Turing Instruction Set

Opcode	Description
Floating Point Instructions	
FADD	FP32 Add
FADD32I	FP32 Add
FCHK	Floating-point Range Check
FFMA32I	FP32 Fused Multiply and Add
FFMA	FP32 Fused Multiply and Add
FMNMX	FP32 Minimum/Maximum
FMUL	FP32 Multiply
FMUL32I	FP32 Multiply

continues on next

Table 1 – continued from previous page

Opcode	Description
FSEL	Floating Point Select
FSET	FP32 Compare And Set
FSETP	FP32 Compare And Set Predicate
FSWZADD	FP32 Swizzle Add
MUFU	FP32 Multi Function Operation
HADD2	FP16 Add
HADD2_32I	FP16 Add
HFMA2	FP16 Fused Mutiply Add
HFMA2_32I	FP16 Fused Mutiply Add
HMMA	Matrix Multiply and Accumulate
HMUL2	FP16 Multiply
HMUL2_32I	FP16 Multiply
HSET2	FP16 Compare And Set
HSETP2	FP16 Compare And Set Predicate
DADD	FP64 Add
DFMA	FP64 Fused Mutiply Add
DMUL	FP64 Multiply
DSETP	FP64 Compare And Set Predicate
Integer Instructions	
BMMA	Bit Matrix Multiply and Accumulate
BMSK	Bitfield Mask
BREV	Bit Reverse
FLO	Find Leading One
IABS	Integer Absolute Value
IADD	Integer Addition
IADD3	3-input Integer Addition
IADD32I	Integer Addition
IDP	Integer Dot Product and Accumulate
IDP4A	Integer Dot Product and Accumulate
IMAD	Integer Multiply And Add
IMMA	Integer Matrix Multiply and Accumulate
IMNMX	Integer Minimum/Maximum

continues on next

Table 1 – continued from previous page

Opcode	Description
IMUL	Integer Multiply
IMUL32I	Integer Multiply
ISCADD	Scaled Integer Addition
ISCADD32I	Scaled Integer Addition
ISETP	Integer Compare And Set Predicate
LEA	LOAD Effective Address
LOP	Logic Operation
LOP3	Logic Operation
LOP32I	Logic Operation
POPC	Population count
SHF	Funnel Shift
SHL	Shift Left
SHR	Shift Right
VABSDIFF	Absolute Difference
VABSDIFF4	Absolute Difference
Conversion Instructions	
F2F	Floating Point To Floating Point Conversion
F2I	Floating Point To Integer Conversion
I2F	Integer To Floating Point Conversion
I2I	Integer To Integer Conversion
I2IP	Integer To Integer Conversion and Packing
FRND	Round To Integer
Movement Instructions	
MOV	Move
MOV32I	Move
MOVM	Move Matrix with Transposition or Expansion
PRMT	Permute Register Pair
SEL	Select Source with Predicate
SGXT	Sign Extend
SHFL	Warp Wide Register Shuffle
Predicate Instructions	
PLOP3	Predicate Logic Operation

continues on next

Table 1 – continued from previous page

Opcode	Description
PSETP	Combine Predicates and Set Predicate
P2R	Move Predicate Register To Register
R2P	Move Register To Predicate Register
Load/Store Instructions	
LD	Load from generic Memory
LDC	Load Constant
LDG	Load from Global Memory
LDL	Load within Local Memory Window
LDS	Load within Shared Memory Window
LDSM	Load Matrix from Shared Memory with Element Size Expansion
ST	Store to Generic Memory
STG	Store to Global Memory
STL	Store to Local Memory
STS	Store to Shared Memory
MATCH	Match Register Values Across Thread Group
QSPC	Query Space
ATOM	Atomic Operation on Generic Memory
ATOMS	Atomic Operation on Shared Memory
ATOMG	Atomic Operation on Global Memory
RED	Reduction Operation on Generic Memory
CCTL	Cache Control
CCTLL	Cache Control
ERRBAR	Error Barrier
MEMBAR	Memory Barrier
CCTLT	Texture Cache Control
Uniform Datapath Instructions	
R2UR	Move from Vector Register to a Uniform Register
S2UR	Move Special Register to Uniform Register
UBMSK	Uniform Bitfield Mask
UBREV	Uniform Bit Reverse
UCLEA	Load Effective Address for a Constant
UFLO	Uniform Find Leading One

continues on next

Table 1 – continued from previous page

Opcode	Description
UIADD3	Uniform Integer Addition
UIADD3.64	Uniform Integer Addition
UIMAD	Uniform Integer Multiplication
UISETP	Integer Compare and Set Uniform Predicate
ULDC	Load from Constant Memory into a Uniform Register
ULEA	Uniform Load Effective Address
ULOP	Logic Operation
ULOP3	Logic Operation
ULOP32I	Logic Operation
UMOV	Uniform Move
UP2UR	Uniform Predicate to Uniform Register
UPLOP3	Uniform Predicate Logic Operation
UPOPC	Uniform Population Count
UPRMT	Uniform Byte Permute
UPSETP	Uniform Predicate Logic Operation
UR2UP	Uniform Register to Uniform Predicate
USEL	Uniform Select
USGXT	Uniform Sign Extend
USHF	Uniform Funnel Shift
USHL	Uniform Left Shift
USHR	Uniform Right Shift
VOTEU	Voting across SIMD Thread Group with Results in Uniform Destin
Texture Instructions	
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4
TMML	Texture MipMap Level
TXD	Texture Fetch With Derivatives
TXQ	Texture Query
Surface Instructions	
SUATOM	Atomic Op on Surface Memory
SULD	Surface Load

continues on next

Table 1 – continued from previous page

Opcode	Description
SURED	Reduction Op on Surface Memory
SUST	Surface Store
Control Instructions	
BMOV	Move Convergence Barrier State
BPT	BreakPoint/Trap
BRA	Relative Branch
BREAK	Break out of the Specified Convergence Barrier
BRX	Relative Branch Indirect
BRXU	Relative Branch with Uniform Register Based Offset
BSSY	Barrier Set Convergence Synchronization Point
BSYNC	Synchronize Threads on a Convergence Barrier
CALL	Call Function
EXIT	Exit Program
JMP	Absolute Jump
JMX	Absolute Jump Indirect
JMXU	Absolute Jump with Uniform Register Based Offset
KILL	Kill Thread
NANOSLEEP	Suspend Execution
RET	Return From Subroutine
RPCMOV	PC Register Move
RTT	Return From Trap
WARPSYNC	Synchronize Threads in Warp
YIELD	Yield Control
Miscellaneous Instructions	
B2R	Move Barrier To Register
BAR	Barrier Synchronization
CS2R	Move Special Register to Register
DEPBAR	Dependency Barrier
GETLMEMBASE	Get Local Memory Base Address
LEPC	Load Effective PC
NOP	No Operation
PMTRIG	Performance Monitor Trigger

continues on next

Table 1 – continued from previous page

Opcode	Description
R2B	Move Register to Barrier
S2R	Move Special Register to Register
SETCTAID	Set CTA ID
SETLMEMBASE	Set Local Memory Base Address
VOTE	Vote Across SIMD Thread Group

6.2. NVIDIA Ampere GPU and Ada Instruction Set

The NVIDIA Ampere GPU and Ada architectures (Compute Capability 8.0, 8.6, and 8.9) have the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- RX for registers
- URX for uniform registers
- SRX for special system-controlled registers
- PX for predicate registers
- UPX for uniform predicate registers
- c[X][Y] for constant memory

Table 7 lists valid instructions for the NVIDIA Ampere architecrture and Ada GPUs.

Table 2: Table 7. NVIDIA Ampere GPU and Ada Instruction Set

Opcode	Description
Floating Point Instructions	
FADD	FP32 Add
FADD32I	FP32 Add
FCHK	Floating-point Range Check
FFMA32I	FP32 Fused Multiply and Add
FFMA	FP32 Fused Multiply and Add
FMNMX	FP32 Minimum/Maximum
FMUL	FP32 Multiply

continues on next page

Table 2 – continued from previous page

Opcode	Description
FMUL32I	FP32 Multiply
FSEL	Floating Point Select
FSET	FP32 Compare And Set
FSETP	FP32 Compare And Set Predicate
FSWZADD	FP32 Swizzle Add
MUFU	FP32 Multi Function Operation
HADD2	FP16 Add
HADD2_32I	FP16 Add
HFMA2	FP16 Fused Mutiply Add
HFMA2_32I	FP16 Fused Mutiply Add
HMMA	Matrix Multiply and Accumulate
HMNMX2	FP16 Minimum / Maximum
HMUL2	FP16 Multiply
HMUL2_32I	FP16 Multiply
HSET2	FP16 Compare And Set
HSETP2	FP16 Compare And Set Predicate
DADD	FP64 Add
DFMA	FP64 Fused Mutiply Add
DMMA	Matrix Multiply and Accumulate
DMUL	FP64 Multiply
DSETP	FP64 Compare And Set Predicate
Integer Instructions	
BMMA	Bit Matrix Multiply and Accumulate
BMSK	Bitfield Mask
BREV	Bit Reverse
FLO	Find Leading One
IABS	Integer Absolute Value
IADD	Integer Addition
IADD3	3-input Integer Addition
IADD32I	Integer Addition
IDP	Integer Dot Product and Accumulate
IDP4A	Integer Dot Product and Accumulate

continues on next page

Table 2 – continued from previous page

Opcode	Description
IMAD	Integer Multiply And Add
IMMA	Integer Matrix Multiply and Accumulate
IMNMX	Integer Minimum/Maximum
IMUL	Integer Multiply
IMUL32I	Integer Multiply
ISCADD	Scaled Integer Addition
ISCADD32I	Scaled Integer Addition
ISETP	Integer Compare And Set Predicate
LEA	LOAD Effective Address
LOP	Logic Operation
LOP3	Logic Operation
LOP32I	Logic Operation
POPC	Population count
SHF	Funnel Shift
SHL	Shift Left
SHR	Shift Right
VABSDIFF	Absolute Difference
VABSDIFF4	Absolute Difference
Conversion Instructions	
F2F	Floating Point To Floating Point Conversion
F2I	Floating Point To Integer Conversion
I2F	Integer To Floating Point Conversion
I2I	Integer To Integer Conversion
I2IP	Integer To Integer Conversion and Packing
I2FP	Integer to FP32 Convert and Pack
F2IP	FP32 Down-Convert to Integer and Pack
FRND	Round To Integer
Movement Instructions	
MOV	Move
MOV32I	Move
MOVM	Move Matrix with Transposition or Expansion
PRMT	Permute Register Pair

continues on next page

Table 2 – continued from previous page

Opcode	Description
SEL	Select Source with Predicate
SGXT	Sign Extend
SHFL	Warp Wide Register Shuffle
Predicate Instructions	
PLOP3	Predicate Logic Operation
PSETP	Combine Predicates and Set Predicate
P2R	Move Predicate Register To Register
R2P	Move Register To Predicate Register
Load/Store Instructions	
LD	Load from generic Memory
LDC	Load Constant
LDG	Load from Global Memory
LDGDEPBAR	Global Load Dependency Barrier
LDGSTS	Asynchronous Global to Shared Memcopy
LDL	Load within Local Memory Window
LDS	Load within Shared Memory Window
LDSM	Load Matrix from Shared Memory with Element Size Expansion
ST	Store to Generic Memory
STG	Store to Global Memory
STL	Store to Local Memory
STS	Store to Shared Memory
MATCH	Match Register Values Across Thread Group
QSPC	Query Space
ATOM	Atomic Operation on Generic Memory
ATOMS	Atomic Operation on Shared Memory
ATOMG	Atomic Operation on Global Memory
RED	Reduction Operation on Generic Memory
CCTL	Cache Control
CCTLL	Cache Control
ERRBAR	Error Barrier
MEMBAR	Memory Barrier
CCTLT	Texture Cache Control

continues on next page

Table 2 – continued from previous page

Opcode	Description
Uniform Datapath Instructions	
R2UR	Move from Vector Register to a Uniform Register
REDUX	Reduction of a Vector Register into a Uniform Register
S2UR	Move Special Register to Uniform Register
UBMSK	Uniform Bitfield Mask
UBREV	Uniform Bit Reverse
UCLEA	Load Effective Address for a Constant
UF2FP	Uniform FP32 Down-convert and Pack
UFLO	Uniform Find Leading One
UIADD3	Uniform Integer Addition
UIADD3.64	Uniform Integer Addition
UIMAD	Uniform Integer Multiplication
UISETP	Integer Compare and Set Uniform Predicate
ULDC	Load from Constant Memory into a Uniform Register
ULEA	Uniform Load Effective Address
ULOP	Logic Operation
ULOP3	Logic Operation
ULOP32I	Logic Operation
UMOV	Uniform Move
UP2UR	Uniform Predicate to Uniform Register
UPLOP3	Uniform Predicate Logic Operation
UPOPC	Uniform Population Count
UPRMT	Uniform Byte Permute
UPSETP	Uniform Predicate Logic Operation
UR2UP	Uniform Register to Uniform Predicate
USEL	Uniform Select
USGXT	Uniform Sign Extend
USHF	Uniform Funnel Shift
USHL	Uniform Left Shift
USHR	Uniform Right Shift
VOTEU	Voting across SIMD Thread Group with Results in Uniform Destination
Texture Instructions	

continues on next page

Table 2 – continued from previous page

Opcode	Description
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4
TMML	Texture MipMap Level
TXD	Texture Fetch With Derivatives
TXQ	Texture Query
Surface Instructions	
SUATOM	Atomic Op on Surface Memory
SULD	Surface Load
SURED	Reduction Op on Surface Memory
SUST	Surface Store
Control Instructions	
BMOV	Move Convergence Barrier State
BPT	BreakPoint/Trap
BRA	Relative Branch
BREAK	Break out of the Specified Convergence Barrier
BRX	Relative Branch Indirect
BRXU	Relative Branch with Uniform Register Based Offset
BSSY	Barrier Set Convergence Synchronization Point
BSYNC	Synchronize Threads on a Convergence Barrier
CALL	Call Function
EXIT	Exit Program
JMP	Absolute Jump
JMX	Absolute Jump Indirect
JMXU	Absolute Jump with Uniform Register Based Offset
KILL	Kill Thread
NANOSLEEP	Suspend Execution
RET	Return From Subroutine
RPCMOV	PC Register Move
WARPSYNC	Synchronize Threads in Warp
YIELD	Yield Control
Miscellaneous Instructions	

continues on next page

Table 2 – continued from previous page

Opcode	Description
B2R	Move Barrier To Register
BAR	Barrier Synchronization
CS2R	Move Special Register to Register
DEPBAR	Dependency Barrier
GETLMEMBASE	Get Local Memory Base Address
LEPC	Load Effective PC
NOP	No Operation
PMTRIG	Performance Monitor Trigger
S2R	Move Special Register to Register
SETCTAID	Set CTA ID
SETLMEMBASE	Set Local Memory Base Address
VOTE	Vote Across SIMD Thread Group

6.3. Hopper Instruction Set

The Hopper architecture (Compute Capability 9.0) has the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- RX for registers
- URX for uniform registers
- SRX for special system-controlled registers
- PX for predicate registers
- UPX for uniform predicate registers
- c[X][Y] for constant memory
- desc[URX][RY] for memory descriptors
- gdesc[URX] for global memory descriptors

Table 8 lists valid instructions for the Hopper GPUs.

Table 3: Table 8. Hopper Instruction Set

Opcode	Description
Floating Point Instructions	

co

Table 3 – continued from previous page

Opcode	Description
FADD	FP32 Add
FADD32I	FP32 Add
FCHK	Floating-point Range Check
FFMA32I	FP32 Fused Multiply and Add
FFMA	FP32 Fused Multiply and Add
FMNMX	FP32 Minimum/Maximum
FMUL	FP32 Multiply
FMUL32I	FP32 Multiply
FSEL	Floating Point Select
FSET	FP32 Compare And Set
FSETP	FP32 Compare And Set Predicate
FSWZADD	FP32 Swizzle Add
MUFU	FP32 Multi Function Operation
HADD2	FP16 Add
HADD2_32I	FP16 Add
HFMA2	FP16 Fused Mutiply Add
HFMA2_32I	FP16 Fused Mutiply Add
HMMA	Matrix Multiply and Accumulate
HMNMX2	FP16 Minimum / Maximum
HMUL2	FP16 Multiply
HMUL2_32I	FP16 Multiply
HSET2	FP16 Compare And Set
HSETP2	FP16 Compare And Set Predicate
DADD	FP64 Add
DFMA	FP64 Fused Mutiply Add
DMMA	Matrix Multiply and Accumulate
DMUL	FP64 Multiply
DSETP	FP64 Compare And Set Predicate
Integer Instructions	
BMMA	Bit Matrix Multiply and Accumulate
BMSK	Bitfield Mask
BREV	Bit Reverse

co

Table 3 – continued from previous page

Opcode	Description
FLO	Find Leading One
IABS	Integer Absolute Value
IADD	Integer Addition
IADD3	3-input Integer Addition
IADD32I	Integer Addition
IDP	Integer Dot Product and Accumulate
IDP4A	Integer Dot Product and Accumulate
IMAD	Integer Multiply And Add
IMMA	Integer Matrix Multiply and Accumulate
IMNMX	Integer Minimum/Maximum
IMUL	Integer Multiply
IMUL32I	Integer Multiply
ISCADD	Scaled Integer Addition
ISCADD32I	Scaled Integer Addition
ISETP	Integer Compare And Set Predicate
LEA	LOAD Effective Address
LOP	Logic Operation
LOP3	Logic Operation
LOP32I	Logic Operation
POPC	Population count
SHF	Funnel Shift
SHL	Shift Left
SHR	Shift Right
VABSDIFF	Absolute Difference
VABSDIFF4	Absolute Difference
VHMNMX	SIMD FP16 3-Input Minimum / Maximum
VIADD	SIMD Integer Addition
VIADDMNMX	SIMD Integer Addition and Fused Min/Max Comparison
VIMNMX	SIMD Integer Minimum / Maximum
VIMNMX3	SIMD Integer 3-Input Minimum / Maximum
Conversion Instructions	
F2F	Floating Point To Floating Point Conversion

co

Table 3 – continued from previous page

Opcode	Description
F2I	Floating Point To Integer Conversion
I2F	Integer To Floating Point Conversion
I2I	Integer To Integer Conversion
I2IP	Integer To Integer Conversion and Packing
I2FP	Integer to FP32 Convert and Pack
F2IP	FP32 Down-Convert to Integer and Pack
FRND	Round To Integer
Movement Instructions	
MOV	Move
MOV32I	Move
MOVM	Move Matrix with Transposition or Expansion
PRMT	Permute Register Pair
SEL	Select Source with Predicate
SGXT	Sign Extend
SHFL	Warp Wide Register Shuffle
Predicate Instructions	
PLOP3	Predicate Logic Operation
PSETP	Combine Predicates and Set Predicate
P2R	Move Predicate Register To Register
R2P	Move Register To Predicate Register
Load/Store Instructions	
FENCE	Memory Visibility Guarantee for Shared or Global Memory
LD	Load from generic Memory
LDC	Load Constant
LDG	Load from Global Memory
LDGDEPBAR	Global Load Dependency Barrier
LDGMC	Reducing Load
LDGSTS	Asynchronous Global to Shared Memcopy
LDL	Load within Local Memory Window
LDS	Load within Shared Memory Window
LDSM	Load Matrix from Shared Memory with Element Size Expansion
STSM	Store Matrix to Shared Memory

co

Table 3 – continued from previous page

Opcode	Description
ST	Store to Generic Memory
STG	Store to Global Memory
STL	Store to Local Memory
STS	Store to Shared Memory
STAS	Asynchronous Store to Distributed Shared Memory With Explicit
SYNCS	Sync Unit
MATCH	Match Register Values Across Thread Group
QSPC	Query Space
ATOM	Atomic Operation on Generic Memory
ATOMS	Atomic Operation on Shared Memory
ATOMG	Atomic Operation on Global Memory
REDAS	Asynchronous Reduction on Distributed Shared Memory With Ex
REDG	Reduction Operation on Generic Memory
CCTL	Cache Control
CCTLL	Cache Control
ERRBAR	Error Barrier
MEMBAR	Memory Barrier
CCTLT	Texture Cache Control
Uniform Datapath Instructions	
R2UR	Move from Vector Register to a Uniform Register
REDUX	Reduction of a Vector Register into a Uniform Register
S2UR	Move Special Register to Uniform Register
UBMSK	Uniform Bitfield Mask
UBREV	Uniform Bit Reverse
UCGABAR_ARV	CGA Barrier Synchronization
UCGABAR_WAIT	CGA Barrier Synchronization
UCLEA	Load Effective Address for a Constant
UF2FP	Uniform FP32 Down-convert and Pack
UFLO	Uniform Find Leading One
UIADD3	Uniform Integer Addition
UIADD3.64	Uniform Integer Addition
UIMAD	Uniform Integer Multiplication

co

Table 3 – continued from previous page

Opcode	Description
UISETP	Integer Compare and Set Uniform Predicate
ULDC	Load from Constant Memory into a Uniform Register
ULEA	Uniform Load Effective Address
ULEPC	Uniform Load Effective PC
ULOP	Logic Operation
ULOP3	Logic Operation
ULOP32I	Logic Operation
UMOV	Uniform Move
UP2UR	Uniform Predicate to Uniform Register
UPLOP3	Uniform Predicate Logic Operation
UPOPC	Uniform Population Count
UPRMT	Uniform Byte Permute
UPSETP	Uniform Predicate Logic Operation
UR2UP	Uniform Register to Uniform Predicate
USEL	Uniform Select
USETMAXREG	Release, Deallocate and Allocate Registers
USGXT	Uniform Sign Extend
USHF	Uniform Funnel Shift
USHL	Uniform Left Shift
USHR	Uniform Right Shift
VOTEU	Voting across SIMD Thread Group with Results in Uniform Destin
Warpgroup Instructions	
BGMMA	Bit Matrix Multiply and Accumulate Across Warps
HGMMA	Matrix Multiply and Accumulate Across a Warpgroup
IGMMA	Integer Matrix Multiply and Accumulate Across a Warpgroup
QGMMA	FP8 Matrix Multiply and Accumulate Across a Warpgroup
WARPGROUP	Warpgroup Synchronization
WARPGROUPSET	Set Warpgroup Counters
Tensor Memory Access Instructions	
UBLKCP	Bulk Data Copy
UBLKPF	Bulk Data Prefetch
UBLKRED	Bulk Data Copy from Shared Memory with Reduction

co

Table 3 – continued from previous page

Opcode	Description
UTMACCTL	TMA Cache Control
UTMACMDFLUSH	TMA Command Flush
UTMALDG	Tensor Load from Global to Shared Memory
UTMAPF	Tensor Prefetch
UTMAREDG	Tensor Store from Shared to Global Memory with Reduction
UTMASTG	Tensor Store from Shared to Global Memory
Texture Instructions	
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4
TMML	Texture MipMap Level
TXD	Texture Fetch With Derivatives
TXQ	Texture Query
Surface Instructions	
SUATOM	Atomic Op on Surface Memory
SULD	Surface Load
SURED	Reduction Op on Surface Memory
SUST	Surface Store
Control Instructions	
ACQBULK	Wait for Bulk Release Status Warp State
BMOV	Move Convergence Barrier State
BPT	BreakPoint/Trap
BRA	Relative Branch
BREAK	Break out of the Specified Convergence Barrier
BRX	Relative Branch Indirect
BRXU	Relative Branch with Uniform Register Based Offset
BSSY	Barrier Set Convergence Synchronization Point
BSYNC	Synchronize Threads on a Convergence Barrier
CALL	Call Function
CGAERRBAR	CGA Error Barrier
ELECT	Elect a Leader Thread
ENDCOLLECTIVE	Reset the MCOLLECTIVE mask

co

Table 3 – continued from previous page

Opcode	Description
EXIT	Exit Program
JMP	Absolute Jump
JMX	Absolute Jump Indirect
JMXU	Absolute Jump with Uniform Register Based Offset
KILL	Kill Thread
NANOSLEEP	Suspend Execution
PREEXIT	Dependent Task Launch Hint
RET	Return From Subroutine
RPCMOV	PC Register Move
WARPSYNC	Synchronize Threads in Warp
YIELD	Yield Control
Miscellaneous Instructions	
B2R	Move Barrier To Register
BAR	Barrier Synchronization
CS2R	Move Special Register to Register
DEPBAR	Dependency Barrier
GETLMEMBASE	Get Local Memory Base Address
LEPC	Load Effective PC
NOP	No Operation
PMTRIG	Performance Monitor Trigger
S2R	Move Special Register to Register
SETCTAID	Set CTA ID
SETLMEMBASE	Set Local Memory Base Address
VOTE	Vote Across SIMT Thread Group

6.4. Blackwell Instruction Set

The Blackwell architecture (Compute Capability 10.0 and 12.0) has the following instruction set format:

```
(instruction) (destination) (source1), (source2) ...
```

Valid destination and source locations include:

- RX for registers
- URX for uniform registers

- SRX for special system-controlled registers
- PX for predicate registers
- UPX for uniform predicate registers
- c[X][Y] for constant memory
- desc[URX][RY] for memory descriptors
- gdesc[URX] for global memory descriptors
- tmem[URX] for tensor memory

Table 8 lists valid instructions for the Blackwell GPUs.

Table 4: Table 8. Blackwell Instruction Set

Opcode	Description
Floating Point Instructions	
FADD	FP32 Add
FADD2	FP32 Add
FADD32I	FP32 Add
FCHK	Floating-point Range Check
FFMA32I	FP32 Fused Multiply and Add
FFMA	FP32 Fused Multiply and Add
FFMA2	FP32 Fused Multiply and Add
FHADD	FP32 Addition
FHFMA	FP32 Fused Multiply and Add
FMNMX	FP32 Minimum/Maximum
FMNMX3	3-Input Floating-point Minimum / Maximum
FMUL	FP32 Multiply
FMUL2	FP32 Multiply
FMUL32I	FP32 Multiply
FSEL	Floating Point Select
FSET	FP32 Compare And Set
FSETP	FP32 Compare And Set Predicate
FSWZADD	FP32 Swizzle Add
MUFU	FP32 Multi Function Operation
HADD2	FP16 Add
HADD2_32I	FP16 Add
HFMA2	FP16 Fused Mutiply Add
HFMA2_32I	FP16 Fused Mutiply Add

continues on next

Table 4 – continued from previous page

Opcode	Description
HMMA	Matrix Multiply and Accumulate
HMNMX2	FP16 Minimum / Maximum
HMUL2	FP16 Multiply
HMUL2_32I	FP16 Multiply
HSET2	FP16 Compare And Set
HSETP2	FP16 Compare And Set Predicate
DADD	FP64 Add
DFMA	FP64 Fused Mutiply Add
DMMA	Matrix Multiply and Accumulate
DMUL	FP64 Multiply
DSETP	FP64 Compare And Set Predicate
OMMA	FP4 Matrix Multiply and Accumulate Across a Warp
QMMA	FP8 Matrix Multiply and Accumulate Across a Warp
Integer Instructions	
BMSK	Bitfield Mask
BREV	Bit Reverse
FLO	Find Leading One
IABS	Integer Absolute Value
IADD	Integer Addition
IADD3	3-input Integer Addition
IADD32I	Integer Addition
IDP	Integer Dot Product and Accumulate
IDP4A	Integer Dot Product and Accumulate
IMAD	Integer Multiply And Add
IMMA	Integer Matrix Multiply and Accumulate
IMNMX	Integer Minimum/Maximum
IMUL	Integer Multiply
IMUL32I	Integer Multiply
ISCADD	Scaled Integer Addition
ISCADD32I	Scaled Integer Addition
ISETP	Integer Compare And Set Predicate
LEA	LOAD Effective Address

continues on next

Table 4 – continued from previous page

Opcode	Description
LOP	Logic Operation
LOP3	Logic Operation
LOP32I	Logic Operation
POPC	Population count
SHF	Funnel Shift
SHL	Shift Left
SHR	Shift Right
VABSDIFF	Absolute Difference
VABSDIFF4	Absolute Difference
VHMNMX	SIMD FP16 3-Input Minimum / Maximum
VIADD	SIMD Integer Addition
VIADDMNMX	SIMD Integer Addition and Fused Min/Max Comparison
VIMNMX	SIMD Integer Minimum / Maximum
VIMNMX3	SIMD Integer 3-Input Minimum / Maximum
Conversion Instructions	
F2F	Floating Point To Floating Point Conversion
F2I	Floating Point To Integer Conversion
I2F	Integer To Floating Point Conversion
I2I	Integer To Integer Conversion
I2IP	Integer To Integer Conversion and Packing
I2FP	Integer to FP32 Convert and Pack
F2IP	FP32 Down-Convert to Integer and Pack
FRND	Round To Integer
Movement Instructions	
MOV	Move
MOV32I	Move
MOVM	Move Matrix with Transposition or Expansion
PRMT	Permute Register Pair
SEL	Select Source with Predicate
SGXT	Sign Extend
SHFL	Warp Wide Register Shuffle
Predicate Instructions	

continues on next

Table 4 – continued from previous page

Opcode	Description
PLOP3	Predicate Logic Operation
PSETP	Combine Predicates and Set Predicate
P2R	Move Predicate Register To Register
R2P	Move Register To Predicate Register
Load/Store Instructions	
FENCE	Memory Visibility Guarantee for Shared or Global Memory
LD	Load from generic Memory
LDC	Load Constant
LDG	Load from Global Memory
LDGDEPBAR	Global Load Dependency Barrier
LDGMC	Reducing Load
LDGSTS	Asynchronous Global to Shared Memcopy
LDL	Load within Local Memory Window
LDS	Load within Shared Memory Window
LDSM	Load Matrix from Shared Memory with Element Size Expansion
STSM	Store Matrix to Shared Memory
ST	Store to Generic Memory
STG	Store to Global Memory
STL	Store to Local Memory
STS	Store to Shared Memory
STAS	Asynchronous Store to Distributed Shared Memory With Explicit Synchroniz
SYNCS	Sync Unit
MATCH	Match Register Values Across Thread Group
QSPC	Query Space
ATOM	Atomic Operation on Generic Memory
ATOMS	Atomic Operation on Shared Memory
ATOMG	Atomic Operation on Global Memory
REDAS	Asynchronous Reduction on Distributed Shared Memory With Explicit Synchronization
REDG	Reduction Operation on Generic Memory
CCTL	Cache Control
CCTLL	Cache Control

continues on next

Table 4 – continued from previous page

Opcode	Description
ERRBAR	Error Barrier
MEMBAR	Memory Barrier
CCTLT	Texture Cache Control
Uniform Datapath Instructions	
CREDUX	Coupled Reduction of a Vector Register into a Uniform Register
CS2UR	Load a Value from Constant Memory into a Uniform Register
LDCU	Load a Value from Constant Memory into a Uniform Register
R2UR	Move from Vector Register to a Uniform Register
REDUX	Reduction of a Vector Register into a Uniform Register
S2UR	Move Special Register to Uniform Register
UBMSK	Uniform Bitfield Mask
UBREV	Uniform Bit Reverse
UCGABAR_ARV	CGA Barrier Synchronization
UCGABAR_WAIT	CGA Barrier Synchronization
UCLEA	Load Effective Address for a Constant
UFADD	Uniform Uniform FP32 Addition
UF2F	Uniform Float-to-Float Conversion
UF2FP	Uniform FP32 Down-convert and Pack
UF2I	Uniform Float-to-Integer Conversion
UF2IP	Uniform FP32 Down-Convert to Integer and Pack
UFFMA	Uniform FP32 Fused Multiply-Add
UFLO	Uniform Find Leading One
UFMNMX	Uniform Floating-point Minimum / Maximum
UFMUL	Uniform FP32 Multiply
UFRND	Uniform Round to Integer
UFSEL	Uniform Floating-Point Select
UFSET	Uniform Floating-Point Compare and Set
UFSETP	Uniform Floating-Point Compare and Set Predicate
UI2F	Uniform Integer to Float conversion
UI2FP	Uniform Integer to FP32 Convert and Pack
UI2I	Uniform Saturating Integer-to-Integer Conversion
UI2IP	Uniform Dual Saturating Integer-to-Integer Conversion and Packing

continues on next

Table 4 – continued from previous page

Opcode	Description
UIABS	Uniform Integer Absolute Value
UIMNMX	Uniform Integer Minimum / Maximum
UIADD3	Uniform Integer Addition
UIADD3.64	Uniform Integer Addition
UIMAD	Uniform Integer Multiplication
UISETP	Uniform Integer Compare and Set Uniform Predicate
ULEA	Uniform Load Effective Address
ULEPC	Uniform Load Effective PC
ULOP	Uniform Logic Operation
ULOP3	Uniform Logic Operation
ULOP32I	Uniform Logic Operation
UMOV	Uniform Move
UP2UR	Uniform Predicate to Uniform Register
UPLOP3	Uniform Predicate Logic Operation
UPOPC	Uniform Population Count
UPRMT	Uniform Byte Permute
UPSETP	Uniform Predicate Logic Operation
UR2UP	Uniform Register to Uniform Predicate
USEL	Uniform Select
USETMAXREG	Release, Deallocate and Allocate Registers
USGXT	Uniform Sign Extend
USHF	Uniform Funnel Shift
USHL	Uniform Left Shift
USHR	Uniform Right Shift
UGETNEXTWORKID	Uniform Get Next Work ID
UMEMSETS	Initialize Shared Memory
UREDGR	Uniform Reduction on Global Memory with Release
USTGR	Uniform Store to Global Memory with Release
UVIADD	Uniform SIMD Integer Addition
UVIMNMX	Uniform SIMD Integer Minimum / Maximum
UVIRTCOUNT	Virtual Resource Management
VOTEU	Voting across SIMD Thread Group with Results in Uniform Destination

continues on next

Table 4 – continued from previous page

Opcode	Description
Tensor Memory Access Instructions	
UBLKCP	Bulk Data Copy
UBLKPF	Bulk Data Prefetch
UBLKRED	Bulk Data Copy from Shared Memory with Reduction
UTMACCTL	TMA Cache Control
UTMACMDFLUSH	TMA Command Flush
UTMALDG	Tensor Load from Global to Shared Memory
UTMAPF	Tensor Prefetch
UTMAREDG	Tensor Store from Shared to Global Memory with Reduction
UTMASTG	Tensor Store from Shared to Global Memory
Tensor Core Memory Instructions	
LDT	Load Matrix from Tensor Memory to Register File
LDTM	Load Matrix from Tensor Memory to Register File
STT	Store Matrix to Tensor Memory from Register File
STTM	Store Matrix to Tensor Memory from Register File
UTCATOMSWS	Perform Atomic operation on SW State Register
UTCBAR	Tensor Core Barrier
UTCCP	Asynchonous data copy from Shared Memory to Tensor Memory
UTCHMMA	Uniform Matrix Multiply and Accumulate
UTCIMMA	Uniform Matrix Multiply and Accumulate
UTCOMMA	Uniform Matrix Multiply and Accumulate
UTCQMMA	Uniform Matrix Multiply and Accumulate
UTCSHIFT	Shift elements in Tensor Memory
Texture Instructions	
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4
TMML	Texture MipMap Level
TXD	Texture Fetch With Derivatives
TXQ	Texture Query
Surface Instructions	
SUATOM	Atomic Op on Surface Memory

continues on next

Table 4 – continued from previous page

Opcode	Description
SULD	Surface Load
SURED	Reduction Op on Surface Memory
SUST	Surface Store
Control Instructions	
ACQBULK	Wait for Bulk Release Status Warp State
ACQSHMINIT	Wait for Shared Memory Initialization Release Status Warp State
BMOV	Move Convergence Barrier State
BPT	BreakPoint/Trap
BRA	Relative Branch
BREAK	Break out of the Specified Convergence Barrier
BRX	Relative Branch Indirect
BRXU	Relative Branch with Uniform Register Based Offset
BSSY	Barrier Set Convergence Synchronization Point
BSYNC	Synchronize Threads on a Convergence Barrier
CALL	Call Function
CGAERRBAR	CGA Error Barrier
ELECT	Elect a Leader Thread
ENDCOLLECTIVE	Reset the MCOLLECTIVE mask
EXIT	Exit Program
JMP	Absolute Jump
JMX	Absolute Jump Indirect
JMXU	Absolute Jump with Uniform Register Based Offset
KILL	Kill Thread
NANOSLEEP	Suspend Execution
PREEXIT	Dependent Task Launch Hint
RET	Return From Subroutine
RPCMOV	PC Register Move
WARPSYNC	Synchronize Threads in Warp
YIELD	Yield Control
Miscellaneous Instructions	
B2R	Move Barrier To Register
BAR	Barrier Synchronization

continues on next

Table 4 – continued from previous page

Opcode	Description
CS2R	Move Special Register to Register
DEPBAR	Dependency Barrier
GETLMEMBASE	Get Local Memory Base Address
LEPC	Load Effective PC
NOP	No Operation
PMTRIG	Performance Monitor Trigger
S2R	Move Special Register to Register
SETCTAID	Set CTA ID
SETLMEMBASE	Set Local Memory Base Address
VOTE	Vote Across SIMT Thread Group

Chapter 7. cu++filt

cu++filt decodes (demangles) low-level identifiers that have been mangled by CUDA C++ into user readable names. For every input alphanumeric word, the output of `cu++filt` is either the demangled name if the name decodes to a CUDA C++ name, or the original name itself.

7.1. Usage

`cu++filt` accepts one or more alphanumeric words (consisting of letters, digits, underscores, dollars, or periods) and attepts to decipher them. The basic usage is as following:

```
cu++filt [options] <symbol(s)>
```

To demangle an entire file, like a binary, pipe the contents of the file to cu++filt, such as in the following command:

```
nm <input file> | cu++filt
```

To demangle function names without printing their parameter types, use the following command :

```
cu++filt -p <symbol(s)>
```

To skip a leading underscore from mangled symbols, use the following command:

```
cu++filt -_ <symbol(s)>
```

Here's a sample output of `cu++filt`:

```
$ cu++filt _Z1fIiEbl
bool f<int>(long)
```

As shown in the output, the symbol `_Z1fIiEbl` was successfully demangled.

To strip all types in the function signature and parameters, use the `-p` option:

```
$ cu++filt -p _Z1fIiEbl
f<int>
```

To skip a leading underscore from a mangled symbol, use the `-_` option:

```
$ cu++filt -_ __Z1fIiEbl
bool f<int>(long)
```

To demangle an entire file, pipe the contents of the file to cu++filt:

```
$ nm test.cubin | cu++filt
0000000000000000 t hello(char *)
0000000000000070 t hello(char *)::display()
0000000000000000 T hello(int *)
```

Symbols that cannot be demangled are printed back to stdout as is:

```
$ cu++filt _ZD2
_ZD2
```

Multiple symbols can be demangled from the command line:

```
$ cu++filt _ZN6Scope15Func1Enez _Z3fooIiPFYneEiEvv _ZD2
Scope1::Func1(__int128, long double, ...)
void foo<int, __int128 (*)(long double), int>()
_ZD2
```

7.2. Command-line Options

Table 9 contains supported command-line options of `cu++filt`, along with a description of what each option does.

Table 1: Table 9. `cu++filt` Command-line Options

Option	Description
-_	Strip underscore. On some systems, the CUDA compiler puts an underscore in front of every name. This option removes the initial underscore. Whether cu++filt removes the underscore by default is target dependent.
-p	When demangling the name of a function, do not display the types of the function's parameters.
-h	Print a summary of the options to cu++filt and exit.
-v	Print the version information of this tool.

7.3. Library Availability

`cu++filt` is also available as a static library (libcufilt) that can be linked against an existing project. The following interface describes it's usage:

```
char* __cu_demangle(const char *id, char *output_buffer, size_t *length, int *status)
```

This interface can be found in the file "nv_decode.h" located in the SDK.

Input Parameters

id Input mangled string.

output_buffer Pointer to where the demangled buffer will be stored. This memory must be allocated with malloc. If output-buffer is NULL, memory will be malloc'd to store the demangled name and returned through the function return value. If the output-buffer is too small, it is expanded using realloc.

length It is necessary to provide the size of the output buffer if the user is providing pre-allocated memory. This is needed by the demangler in case the size needs to be reallocated. If the length is non-null, the length of the demangled buffer is placed in length.

status *status is set to one of the following values:

- 0 - The demangling operation succeeded
- -1 - A memory allocation failure occurred
- -2 - Not a valid mangled id
- -3 - An input validation failure has occurred (one or more arguments are invalid)

Return Value

A pointer to the start of the NUL-terminated demangled name, or NULL if the demangling fails. The caller is responsible for deallocating this memory using free.

Note: This function is thread-safe.

Example Usage

```
#include <stdio.h>
#include <stdlib.h>
#include "nv_decode.h"

int main()
{
  int     status;
  const char *real_mangled_name="_ZN8clstmp01I5cls01E13clstmp01_mf01Ev";
  const char *fake_mangled_name="B@d_iDentiFier";

  char* realname = __cu_demangle(fake_mangled_name, 0, 0, &status);
  printf("fake_mangled_name:\t result => %s\t status => %d\n", realname, status);
  free(realname);

  size_t size = sizeof(char)*1000;
  realname = (char*)malloc(size);
  __cu_demangle(real_mangled_name, realname, &size, &status);
  printf("real_mangled_name:\t result => %s\t status => %d\n", realname, status);
  free(realname);

  return 0;
}
```

This prints:

```
fake_mangled_name:   result => (null)     status => -2
real_mangled_name:   result => clstmp01<cls01>::clstmp01_mf01()   status => 0
```

Chapter 8. nvprune

`nvprune` prunes host object files and libraries to only contain device code for the specified targets.

8.1. Usage

`nvprune` accepts a single input file each time it's run, emitting a new output file. The basic usage is as following:

```
nvprune [options] -o <outfile> <infile>
```

The input file must be either a relocatable host object or static library (not a host executable), and the output file will be the same format.

Either the –arch or –generate-code option must be used to specify the target(s) to keep. All other device code is discarded from the file. The targets can be either a sm_NN arch (cubin) or compute_NN arch (ptx).

For example, the following will prune libcublas_static.a to only contain sm_120 cubin rather than all the targets which normally exist:

```
nvprune -arch sm_120 libcublas_static.a -o libcublas_static120.a
```

Note that this means that libcublas_static120.a will not run on any other architecture, so should only be used when you are building for a single architecture.

8.2. Command-line Options

Table 10 contains supported command-line options of `nvprune`, along with a description of what each option does. Each option has a long name and a short name, which can be used interchangeably.

Table 1: Table 10. nvprune Command-line Options

Option (long)	Option (short)	Description
--arch <gpu architecture name>,...	-arch	Specify the name of the NVIDIA GPU architecture which will remain in the object or library.
--generate-code	-gencode	This option is same format as nvcc –generate-code option, and provides a way to specify multiple architectures which should remain in the object or library. Only the 'code' values are used as targets to match. Allowed keywords for this option: 'arch','code'.
--no-relocatable-elf	-no-relocatable-elf	Don't keep any relocatable ELF.
--output-file	-o	Specify name and location of the output file.
--help	-h	Print this help information on this tool.
--options-file <file>,...	-optf	Include command line options from specified file.
--version	-V	Print version information on this tool.

Chapter 9. Appendix

9.1. JSON Format

The output of nvdisasm is human-readable text which is not formatted for machine consumption. Any tool consuming the output of nvdisasm must parse the human-readable text which can be slow and any minor changes to the text can break the parser.

JSON-based format provides an efficient and extensible method to output machine readable data from nvdisasm. The option -json can be used to produce a JSON document that adheres to the following JSON schema definition.

```
{
    "$id": "https://nvidia.com/cuda/cuda-binary-utilities/index.html#json-format",
    "description": "A JSON schema for NVIDIA CUDA disassembler. The $id attribute is
↪not a real URL but a unique identifier for the schema",
    "$schema": "https://json-schema.org/draft/2020-12/schema",
    "title": "A JSON schema for NVIDIA CUDA disassembler",
    "version": "13-1-0",
    "type": "array",
    "minItems": 2,
    "prefixItems": [
        {
            "$ref": "#/$defs/metadata"
        },
        {
            "description": "A list of CUDA functions",
            "type": "array",
            "minItems": 1,
            "items": {
                "$ref": "#/$defs/function"
            }
        }
    ],
    "$defs": {
        "metadata": {
            "type": "object",
            "properties": {
                "ELF": {
                    "$ref": "#/$defs/elf-metadata"
                },
                "SM": {
                    "type": "object",
                    "properties": {
```

(continues on next page)

(continued from previous page)

```
                    "version": {
                        "$ref": "#/$defs/sm-version"
                    }
                },
                "required": [
                    "version"
                ]
            },
            "SchemaVersion": {
                "$ref": "#/$defs/version"
            },
            "Producer": {
                "type": "string",
                "description": "Name and version of the CUDA disassembler tool",
                "maxLength": 1024
            },
            "Description": {
                "type": "string",
                "description": "A description that may be empty",
                "maxLength": 1024
            },
            ".note.nv.tkinfo": {
                "$ref": "#/$defs/Elf64_NV_TKinfo"
            }
        },
        "required": [
            "ELF",
            "SM",
            "SchemaVersion",
            "Producer",
            "Description"
        ]
    },
    "elf-metadata": {
        "type": "object",
        "properties": {
            "layout-id": {
                "description": "Indicates the layout of the ELF file, part of the
↪ELF header flags. Undocumented enum",
                "type": "integer"
            },
            "ei_osabi": {
                "description": "Operating system/ABI identification",
                "type": "integer"
            },
            "ei_abiversion": {
                "description": "ABI version",
                "type": "integer"
            }
        },
        "required": [
            "layout-id",
            "ei_osabi",
            "ei_abiversion"
        ]
    },
```

(continues on next page)

(continued from previous page)

```
        "sm-version": {
            "type": "object",
            "properties": {
                "major": {
                    "type": "integer"
                },
                "minor": {
                    "type": "integer"
                }
            },
            "required": [
                "major",
                "minor"
            ]
        },
        "version": {
            "type": "object",
            "properties": {
                "major": {
                    "type": "integer"
                },
                "minor": {
                    "type": "integer"
                },
                "revision": {
                    "type": "integer"
                }
            },
            "required": [
                "major",
                "minor",
                "revision"
            ]
        },
        "sass-instruction-attribute": {
            "type": "object",
            "additionalProperties": {
                "type": "string"
            }
        },
        "sass-instruction": {
            "type": "object",
            "properties": {
                "predicate": {
                    "type": "string",
                    "description": "Instruction predicate"
                },
                "opcode": {
                    "type": "string",
                    "description": "The instruction opcode. May be empty to indicate
↪a gap between non-contiguous instructions"
                },
                "operands": {
                    "type": "string",
                    "description": "Instruction operands separated by commas"
                },
```

(continues on next page)

(continued from previous page)

```
                "extra": {
                    "type": "string",
                    "description": "Optional field"
                },
                "other-attributes": {
                    "type": "object",
                    "description": "Additional instruction attributes encoded as a
↪map of string:string key-value pairs. Example: {'control-flow': 'True'}",
                    "properties": {
                        "control-flow": {
                            "const": ["True"],
                            "description": "True if the instruction is a control flow
↪instruction"
                        },
                        "subroutine-call": {
                            "const": ["True"],
                            "description": "True if the instruction is a subroutine
↪call"
                        }
                    }
                },
                "other-flags": {
                    "type": "array",
                    "description": "Aditional instruction attributes encoded as a
↪list strings",
                    "items": {
                        "type": "string"
                    }
                }
            },
            "required": [
                "opcode"
            ]
        },
        "function": {
            "type": "object",
            "properties": {
                "function-name": {
                    "type": "string"
                },
                "start": {
                    "type": "integer",
                    "description": "The function's start virtual address"
                },
                "length": {
                    "type": "integer",
                    "description": "The function's length in bytes"
                },
                "other-attributes": {
                    "type": "array",
                    "items": {
                        "type": "string"
                    }
                },
                "sass-instructions": {
                    "type": "array",
```

(continues on next page)

(continued from previous page)

```
                    "items": {
                        "$ref": "#/$defs/sass-instruction"
                    }
                }
            },
            "required": [
                "function-name",
                "start",
                "length",
                "sass-instructions"
            ]
        },
        "Elf64_NV_TKinfo": {
            "type": "object",
            "properties": {
                "tki_toolkitVersion": {
                    "type": "integer"
                },
                "tki_objFname": {
                    "type": "string"
                },
                "tki_toolName": {
                    "type": "string"
                },
                "tki_toolVersion": {
                    "type": "string"
                },
                "tki_toolBranch": {
                    "type": "string"
                },
                "tki_toolOptions": {
                    "type": "string"
                }
            },
            "required": [
                "tki_toolkitVersion",
                "tki_objFname",
                "tki_toolName",
                "tki_toolVersion",
                "tki_toolBranch",
                "tki_toolOptions"
            ]
        }
    }
}
```

Notes about sass-instruction objects

- The `other-attributes` object may contain `"control-flow": "True"` key-pair to indicate control flow instructions and `"subroutine-call": "True"` key-pair to indicate subroutine call instructions.
- The address of the nth (0-based) SASS instruction can be computed as start + n * instruction size . The instruction size is 16 bytes.
- The JSON list may contain empty instruction objects; these objects count towards the instruction index, as they are indicating gaps between non-contiguous instructions.

- An empty instruction object has the single field `opcode` with an empty string value : `"opcode" : ""`

Here's a sample output from `nvdisasm -json`

```
[
    // First element in the list: Metadata
    {
        // ELF Metadata
        "ELF": {
            "layout-id": 4,
            "ei_osabi": 51,
            "ei_abiversion": 7
        },
        // SASS code SM version: SM89 (16 bytes instructions)
        "SM": {
            "version": {
                "major": 8,
                "minor": 9
            }
        },
        "SchemaVersion": {
            "major": 12,
            "minor": 8,
            "revision": 0
        },
        // Release details of nvdisasm
        "Producer": "nvdisasm V12.8.14 Build r570_00.r12.8/compiler.35033008_0",
        "Description": ""
    },
    // Second element in the list: Functions
    [
        {
            "function-name": "_Z10exampleKernelv",
            // Function start address
            "start": 0,
            // Function length in bytes
            "length": 384,
            "other-attributes": [],
            // SASS instructions
            "sass-instructions": [
                {
                    // Instruction at 0x00
                    "opcode": "IMAD.MOV.U32",
                    "operands": "R1,RZ,RZ,c[0x0][0x28]"
                },
                {
                    // Instruction at 0x10 (16 bytes increment)
                    "opcode": "MOV",
                    "operands": "R0,0x0"
                },
                {
                    // Instruction at 0x20
                    "opcode": "IMAD.MOV.U32",
                    "operands": "R4,RZ,RZ,c[0x4][0x8]"
                },
                // [...]
                {
```

(continues on next page)

(continued from previous page)

```
                    "opcode": "CALL.ABS.NOINC",
                    "operands": "R2",
                    // other-attributes is an optional that can indicate control flow
↪instructions
                    "other-attributes": {
                        "control-flow": "True",
                        "subroutine-call": "True"
                    }
                },
                {
                    "opcode": "EXIT",
                    "other-attributes": {
                        "control-flow": "True"
                    }
                },
                {
                    "opcode": "NOP"
                }
            ]
        }
    ]
]
```

Chapter 10. Notices

10.1. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND 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. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

10.2. OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

10.3. Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright