CUDA Binary Utilities

Application Note

DA-06762-001_v11.1 | September 2020

Table of Contents

List of Figures

List of Tables

Chapter 1. Overview

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

1.1. 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.

1.2. 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. 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	Yes	No

<table>
<tr><th></th><th>cuobjdump</th><th>nvdisasm</th></tr>
<tr><td>Executables
Object files
Static libraries
External fatbinary files</td><td></td><td></td></tr>
<tr><td>Control flow analysis and output</td><td>No</td><td>Yes</td></tr>
<tr><td>Advanced display options</td><td>No</td><td>Yes</td></tr>
</table>

Chapter 2. 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.

2.1. 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 a.out -sass -ptx
Fatbin elf code:
================
arch = sm_70
code version = [1,7]
producer = cuda
host = linux
compile_size = 64bit
identifier = add.cu

code for sm_70
        Function : _Z3addPiS_S_
.headerflags    @"EF_CUDA_SM70 EF_CUDA_PTX_SM(EF_CUDA_SM70)"
/*0000*/      IMAD.MOV.U32 R1, RZ, RZ, c[0x0][0x28] ;  /* 0x00000a00ff017624 */
                                                       /* 0x000fd000078e00ff */
/*0010*/ @!PT SHFL.IDX PT, RZ, RZ, RZ, RZ ;            /* 0x000000fffffff389 */
                                                       /* 0x000fe200000e00ff */
```

```
/*0020*/      IMAD.MOV.U32 R2, RZ, RZ, c[0x0][0x160] ; /* 0x00005800ff027624 */
                                                       /* 0x000fe200078e00ff */
/*0030*/      MOV R3, c[0x0][0x164] ;                  /* 0x0000590000037a02 */
                                                       /* 0x000fe20000000f00 */
/*0040*/      IMAD.MOV.U32 R4, RZ, RZ, c[0x0][0x168] ; /* 0x00005a00ff047624 */
                                                       /* 0x000fe200078e00ff */
/*0050*/      MOV R5, c[0x0][0x16c] ;                  /* 0x00005b0000057a02 */
                                                       /* 0x000fcc0000000f00 */
/*0060*/      LDG.E.SYS R2, [R2] ;                     /* 0x0000000002027381 */
                                                       /* 0x000ea800001ee900 */
/*0070*/      LDG.E.SYS R5, [R4] ;                     /* 0x0000000004057381 */
                                                       /* 0x000ea200001ee900 */
/*0080*/      IMAD.MOV.U32 R6, RZ, RZ, c[0x0][0x170] ; /* 0x00005c00ff067624 */
                                                       /* 0x000fe200078e00ff */
/*0090*/      MOV R7, c[0x0][0x174] ;                  /* 0x00005d0000077a02 */
                                                       /* 0x000fe40000000f00 */
/*00a0*/      IADD3 R9, R2, R5, RZ ;                   /* 0x0000000502097210 */
                                                       /* 0x004fd00007ffe0ff */
/*00b0*/      STG.E.SYS [R6], R9 ;                     /* 0x0000000906007386 */
                                                       /* 0x000fe2000010e900 */
/*00c0*/      EXIT ;                                   /* 0x000000000000794d */
                                                       /* 0x000fea0003800000 */
/*00d0*/      BRA 0xd0;                                /* 0xfffffff000007947 */
                                                       /* 0x000fc0000383ffff */
/*00e0*/      NOP;                                     /* 0x0000000000007918 */
                                                       /* 0x000fc00000000000 */
/*00f0*/      NOP;                                     /* 0x0000000000007918 */
                                                       /* 0x000fc00000000000 */
        .......................
```

```
Fatbin ptx code:
================
arch = sm_70
code version = [7,0]
producer = cuda
host = linux
compile_size = 64bit
compressed
identifier = add.cu

.version 7.0
.target sm_70
.address_size 64

.visible .entry _Z3addPiS_S_(
.param .u64 _Z3addPiS_S__param_0,
.param .u64 _Z3addPiS_S__param_1,
.param .u64 _Z3addPiS_S__param_2
)
{
.reg .s32 %r<4>;
.reg .s64 %rd<7>;

ld.param.u64 %rd1, [_Z3addPiS_S__param_0];
ld.param.u64 %rd2, [_Z3addPiS_S__param_1];
ld.param.u64 %rd3, [_Z3addPiS_S__param_2];
cvta.to.global.u64 %rd4, %rd3;
cvta.to.global.u64 %rd5, %rd2;
cvta.to.global.u64 %rd6, %rd1;
ld.global.u32 %r1, [%rd6];
ld.global.u32 %r2, [%rd5];
add.s32 %r3, %r2, %r1;
st.global.u32 [%rd4], %r3;
ret;
}
```

As shown in the output, the `a.out` host binary contains cubin and ptx code for sm_70.

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

```
$ cuobjdump a.out -lelf
ELF file    1: add_new.sm_60.cubin
ELF file    2: add_new.sm_70.cubin
ELF file    3: add_old.sm_60.cubin
ELF file    4: add_old.sm_70.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_60.cubin
Extracting ELF file    2: add_new.sm_70.cubin
Extracting ELF file    3: add_old.sm_60.cubin
Extracting ELF file    4: add_old.sm_70.cubin
```

To extract the cubin named `add_new.sm_70.cubin`:

```
$ cuobjdump a.out -xelf add_new.sm_70.cubin
Extracting ELF file    1: add_new.sm_70.cubin
```

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

```
$ cuobjdump a.out -xelf _old
Extracting ELF file    1: add_old.sm_60.cubin
Extracting ELF file    2: add_old.sm_70.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:110 CONSTANT[2]:296 CONSTANT[14]:16
 Function calculate:
  REG:7 STACK:400 SHARED:0 LOCAL:0 CONSTANT[0]:328 TEXTURE:0 SURFACE:0 SAMPLER:0
 Function mysurf_func:
  REG:18 STACK:0 SHARED:0 LOCAL:0 CONSTANT[0]:444 TEXTURE:0 SURFACE:1 SAMPLER:0
 Function mytexsampler_func:
  REG:42 STACK:0 SHARED:0 LOCAL:0 CONSTANT[0]:472 TEXTURE:4 SURFACE:0 SAMPLER:1
 Function mysharedfunc:
  REG:30 STACK:0 SHARED:20 LOCAL:0 CONSTANT[0]:192 CONSTANT[16]:440 TEXTURE:0
 SURFACE:0 SAMPLER:0
```

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

2.2. 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 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.
`--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_30','sm_32','sm_35','sm_37','sm_50','sm_52','sm_53' ,'sm_60','sm_61','sm_62' ,'sm_70','sm_72','sm_75' ,'sm_80'.
`--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.

Option (long)	Option (short)	Description
		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.
`--options-file <file>,...`	`-optf`	Include command line options from specified file.
`--sort-functions`	`-sort`	Sort functions when dumping sass.
`--version`	`-V`	Print version information on this tool.

Chapter 3. 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.

3.1. Usage

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

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

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

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

Here's a sample output of nvdisasm:

```
        .headerflags    @"EF_CUDA_TEXMODE_UNIFIED EF_CUDA_64BIT_ADDRESS EF_CUDA_SM30
 EF_CUDA_PTX_SM(EF_CUDA_SM30) "


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

......

//--------------------- .text._Z4addXPii          --------------------------
        .section        .text._Z4addXPii,"ax",@progbits
        .sectioninfo    @"SHI_REGISTERS=11 "
        .align   4
        .global         _Z4addXPii
        .type           _Z4addXPii,@function
        .size           _Z4addXPii,(.L_19 - _Z4addXPii)
        .other          _Z4addXPii,@"STO_CUDA_ENTRY STV_DEFAULT "
_Z4addXPii:
```

```
.text._Z4addXPii:
        /*0008*/                MOV R1, c[0x0][0x44];
        /*0010*/                ISUB R1, R1, 0x8;
        /*0018*/                MOV R0, c[0x0][0x148];
        /*0020*/                IADD R6.CC, R1, c[0x0][0x24];
        /*0028*/                ISETP.LT.AND P0, PT, R0, 0x1, PT;
        /*0030*/                MOV R8, c[0x0][0x140];
        /*0038*/                MOV R9, c[0x0][0x144];
        /*0048*/                IADD.X R7, RZ, RZ;
        /*0050*/                ISUB R10, R6, c[0x0][0x24];
        /*0058*/            @P0 BRA `(.L_2);
        /*0060*/                LD.E R0, [R8];
        /*0068*/                MOV R2, RZ;
        /*0070*/                NOP;
        /*0078*/                NOP;
.L_3:
        /*0088*/                IADD R2, R2, 0x1;
        /*0090*/                MOV R3, R0;
        /*0098*/                IADD R0, R0, 0x1;
        /*00a0*/                ISETP.LT.AND P0, PT, R2, c[0x0][0x148], PT;
        /*00a8*/            @P0 BRA `(.L_3);
        /*00b0*/                IADD R0, R3, 0x1;
        /*00b8*/                ST.E [R8], R0;
.L_2:
        /*00c8*/                S2R R0, SR_TID.X;
        /*00d0*/                ISETP.NE.AND P0, PT, R0, RZ, PT;
        /*00d8*/            @P0 EXIT ;
        /*00e0*/                LD.E R0, [R8];
        /*00e8*/                MOV R4, c[0x0][0xf0];
        /*00f0*/                MOV R5, c[0x0][0xf4];
        /*00f8*/                STL [R10], R0;
        /*0108*/                JCAL `(vprintf);
        /*0110*/                EXIT ;
.L_4:
        /*0118*/                BRA `(.L_4);
.L_19:


//-------------------- SYMBOLS --------------------------

        .type           vprintf,@function
```

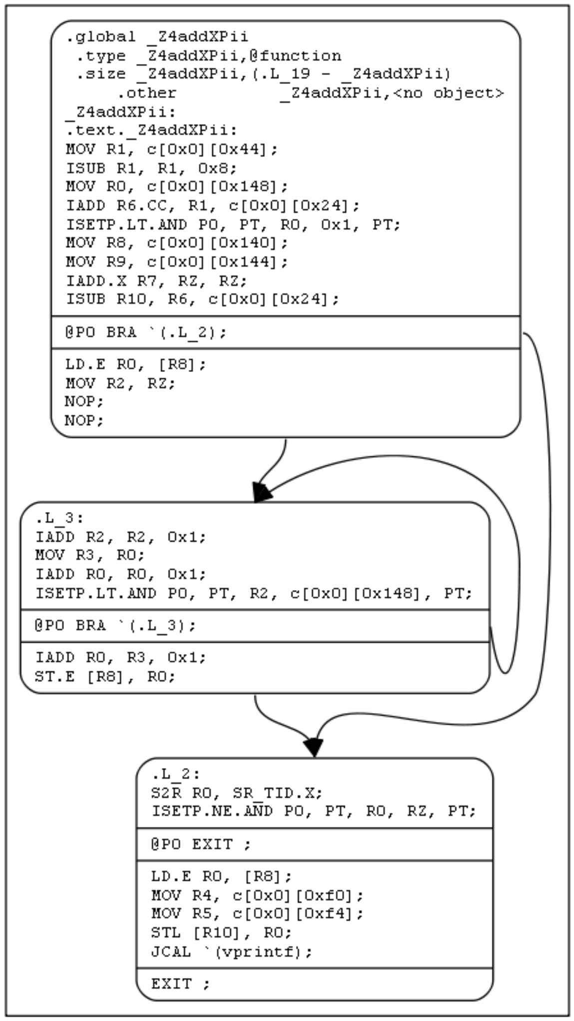
`nvdisasm` is capable of generating control flow of CUDA assembly in the format of DOT graph description language. The output of the control flow from nvdisasm can be directly imported to a DOT graph visualization tool such as Graphviz.

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:

Figure 1. Control Flow Graph



`nvdisasm` is capable of showing the register (CC, 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 (left columns are omitted):

```
                                                          // +------+---------------+-----+
                                                          // | CC   |     GPR       |PRED |
                                                          // |      |    0000000000 |     |
                                                          // | # 01 |  # 0123456789 | # 0 |
                                                          // +------+---------------+-----+
_main10acosParams                                         // |      |               |     |
_main10acosParams,@function                               // |      |               |     |
_main10acosParams,(.L_17 - _Z9acos_main10acosParams)      // |      |               |     |
_main10acosParams,@"STO_CUDA_ENTRY STV_DEFAULT"           // |      |               |     |
                                                          // |      |               |     |
                                                          // |      |               |     |
    MOV R1, c[0x0][0x44];                                 // |      |  1  ^         |     |
    S2R R0, SR_CTAID.X;                                   // |      |  2 ^:         |     |
    S2R R3, SR_TID.X;                                     // |      |  3 :: ^       |     |
    IMAD R3, R0, c[0x0][0x28], R3;                        // |      |  3 v: x       |     |
    MOV R0, c[0x0][0x28];                                 // |      |  3 ^: :       |     |
    ISETP.GE.AND P0, PT, R3, c[0x0][0x150], PT;           // |      |  3 :: v       | 1 ^ |
```

```
    IMUL R0, R0, c[0x0][0x34];                            // |      |  3 x: :       | 1 : |
@P0 EXIT;                                                 // |      |  3 :: :       | 1 v |
    MOV32I R8, 0x4;                                       // |      |  4 :: :    ^  |     |
    MOV32I R9, 0x3c94d2e9;                                // |      |  5 :: :    :^ |     |
    NOP;                                                  // |      |  5 :: :    :: |     |
    NOP;                                                  // |      |  5 :: :    :: |     |
    NOP;                                                  // |      |  5 :: :    :: |     |
    NOP;                                                  // |      |  5 :: :    :: |     |
                                                          // |      |  5 :: :    :: |     |
    IMAD R6.CC, R3, R8, c[0x0][0x140];                    // | 1  ^ |  6 :: v  ^ v: |     |
    IMAD.HI.X R7, R3, R8, c[0x0][0x144];                  // | 1  v |  7 :: v  :^v: |     |
    LD.E R2, [R6];                                        // |      |  8 ::^:  vv:: |     |
    FADD.FTZ R4, -|R2|, 1;                                // |      |  7 ::v:^   :: |     |
    FSETP.GT.FTZ.AND P0, PT, |R2|, c[0x2][0x0], PT;       // |      |  7 ::v::   :: | 1 ^ |
    FMUL.FTZ R4, R4, 0.5;                                 // |      |  7 ::::x   :: | 1 : |
    F2F.FTZ.F32.F32 R5, |R2|;                             // |      |  8 ::v::^  :: | 1 : |
    MUFU.RSQ R4, R4;                                      // |      |  8 ::::x:  :: | 1 : |
@P0 MUFU.RCP R5, R4;                                      // |      |  8 ::::v^  :: | 1 v |
    FMUL.FTZ R4, R5, R5;                                  // |      |  8 ::::^v  :: | 1 : |
    IMAD R6.CC, R3, R8, c[0x0][0x148];                    // | 1  ^ |  9 :::v::^ v: | 1 : |
    FFMA.FTZ R7, R4, c[0x2][0x4], R9;                     // | 1  : | 10 ::::v::^:v | 1 : |
    FFMA.FTZ R7, R7, R4, c[0x2][0x8];                     // | 1  : | 10 ::::v::x:: | 1 : |
    FFMA.FTZ R7, R7, R4, c[0x2][0xc];                     // | 1  : | 10 ::::v::x:: | 1 : |
    FFMA.FTZ R7, R7, R4, c[0x2][0x10];                    // | 1  : | 10 ::::v::x:: | 1 : |
    FMUL.FTZ R4, R7, R4;                                  // | 1  : | 10 ::::x::v:: | 1 : |
    IMAD.HI.X R7, R3, R8, c[0x0][0x14c];                  // | 1  v | 10 :::v:::^v: | 1 : |
    FFMA.FTZ R4, R4, R5, R5;                              // |      | 10 ::::xv:::: | 1 : |
    IADD R3, R3, R0;                                      // |      |  9 v::x: :::: | 1 : |
    FADD32I.FTZ R5, -R4, 1.5707963705062866211;           // |      | 10 ::::v^:::: | 1 : |
@P0 FADD.FTZ R5, R4, R4;                                  // |      | 10 ::::v^:::: | 1 v |
    ISETP.LT.AND P0, PT, R3, c[0x0][0x150], PT;           // |      |  9 :::v ::::: | 1 ^ |
    FADD32I.FTZ R4, -R5, 3.1415927410125732422;           // |      | 10 ::::^v:::: | 1 : |
    FCMP.LT.FTZ R2, R4, R5, R2;                           // |      | 10 ::x:vv:::: | 1 : |
    ST.E [R6], R2;                                        // |      |  8 ::v:  vv:: | 1 : |
@P0 BRA `(.L_1);                                          // |      |  5 :: :    :: | 1 v |
    MOV RZ, RZ;                                           // |      |  1  :         |     |
    EXIT;                                                 // |      |  1  :         |     |
                                                          // +......+...............+.....+
    BRA `(.L_2);                                          // |      |               |     |
                                                          // +------+---------------+-----+
                                                          // Legend:
                                                          //     ^       : Register assignment
                                                          //     v       : Register usage
                                                          //     x       : Register usage and reassignment
                                                          //     :       : Register in use
                                                          //     <space> : Register not in use
                                                          //     #       : Number of occupied registers
```

3.2. 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 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:

Option (long)	Option (short)	Description
		'SM30','SM32','SM35','SM37','SM50','SM52','SM53' ,'SM60','SM61','SM62' ,'SM70','SM72','SM75' ,'SM80'.
`--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.
`--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 in a format consumable by graphviz tools (such as dot).
`--print-code`	`-c`	Only print code sections.
`--print-instr-offsets-cfg`	`-poff`	When specified, print instruction offsets in the control flow graph. This should be used along with the option --output-control-flow-graph.
`--print-instruction-encoding`	`-hex`	When specified, print the encoding bytes after each disassembled operation.
`--print-life-ranges`	`-plr`	Print register life range information in a trailing column in the produced disassembly.
`--print-line-info`	`-g`	Annotate disassembly with source line information obtained from .debug_line section, if present.
`--print-line-info-ptx`	`-gp`	Annotate disassembly with source line information obtained from .nv_debug_line_sass section, if present.
`--print-raw`	`-raw`	Print the disassembly without any attempt to beautify it.
`--separate-functions`	`-sf`	Separate the code corresponding with function symbols by some new lines to let them stand out in the printed disassembly.

Option (long)	Option (short)	Description
`--version`	`-V`	Print version information on this tool.

Chapter 4. Instruction Set Reference

This is an instruction set reference for NVIDIA® GPU architectures Kepler, Maxwell, Pascal, Volta, Turing and Ampere.

4.1. Kepler Instruction Set

The Kepler architecture (Compute Capability 3.x) has the following instruction set format:

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

Valid destination and source locations include:

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

Table 4 lists valid instructions for the Kepler GPUs.

Table 4. Kepler Instruction Set

Opcode	Description
Floating Point Instructions	
FFMA	FP32 Fused Multiply Add
FADD	FP32 Add
FCMP	FP32 Compare
FMUL	FP32 Multiply
FMNMX	FP32 Minimum/Maximum
FSWZ	FP32 Swizzle
FSET	FP32 Set
FSETP	FP32 Set Predicate
FCHK	FP32 Division Test
RRO	FP Range Reduction Operator
MUFU	FP Multi-Function Operator

Opcode	Description
DFMA	FP64 Fused Multiply Add
DADD	FP64 Add
DMUL	FP64 Multiply
DMNMX	FP64 Minimum/Maximum
DSET	FP64 Set
DSETP	FP64 Set Predicate
Integer Instructions	
IMAD	Integer Multiply Add
IMADSP	Integer Extract Multiply Add
IMUL	Integer Multiply
IADD	Integer Add
ISCADD	Integer Scaled Add
ISAD	Integer Sum Of Abs Diff
IMNMX	Integer Minimum/Maximum
BFE	Integer Bit Field Extract
BFI	Integer Bit Field Insert
SHR	Integer Shift Right
SHL	Integer Shift Left
SHF	Integer Funnel Shift
LOP	Integer Logic Op
FLO	Integer Find Leading One
ISET	Integer Set
ISETP	Integer Set Predicate
ICMP	Integer Compare and Select
POPC	Population Count
Conversion Instructions	
F2F	Float to Float
F2I	Float to Integer
I2F	Integer to Float
I2I	Integer to Integer
Movement Instructions	
MOV	Move
SEL	Conditional Select/Move
PRMT	Permute
SHFL	Warp Shuffle
Predicate/CC Instructions	
P2R	Predicate to Register
R2P	Register to Predicate

Opcode	Description
CSET	CC Set
CSETP	CC Set Predicate
PSET	Predicate Set
PSETP	Predicate Set Predicate
Texture Instructions	
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4 Texels
TXQ	Texture Query
Compute Load/Store Instructions	
LDC	Load from Constant
LD	Load from Memory
LDG	Non-coherent Global Memory Load
LDL	Load from Local Memory
LDS	Load from Shared Memory
LDSLK	Load from Shared Memory and Lock
ST	Store to Memory
STL	Store to Local Memory
STS	Store to Shared Memory
STSCUL	Store to Shared Memory Conditionally and Unlock
ATOM	Atomic Memory Operation
RED	Atomic Memory Reduction Operation
CCTL	Cache Control
CCTLL	Cache Control (Local)
MEMBAR	Memory Barrier
Surface Memory Instructions	
SUCLAMP	Surface Clamp
SUBFM	Surface Bit Field Merge
SUEAU	Surface Effective Address
SULDGA	Surface Load Generic Address
SUSTGA	Surface Store Generic Address
Control Instructions	
BRA	Branch to Relative Address
BRX	Branch to Relative Indexed Address
JMP	Jump to Absolute Address
JMX	Jump to Absolute Indexed Address
CAL	Call to Relative Address
JCAL	Call to Absolute Address

Opcode	Description
RET	Return from Call
BRK	Break from Loop
CONT	Continue in Loop
SSY	Set Sync Relative Address
PBK	Pre-Break Relative Address
PCNT	Pre-Continue Relative Address
PRET	Pre-Return Relative Address
BPT	Breakpoint/Trap
EXIT	Exit Program
Miscellaneous Instructions	
NOP	No Operation
S2R	Special Register to Register
B2R	Barrier to Register
BAR	Barrier Synchronization
VOTE	Query condition across threads

4.2. Maxwell and Pascal Instruction Set

The Maxwell (Compute Capability 5.x) and the Pascal (Compute Capability 6.x) architectures have the following instruction set format:

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

Valid destination and source locations include:

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

Table 5 lists valid instructions for the Maxwell and Pascal GPUs.

Table 5. Maxwell and Pascal Instruction Set

Opcode	Description
Floating Point Instructions	
FADD	FP32 Add
FCHK	Single Precision FP Divide Range Check
FCMP	FP32 Compare to Zero and Select Source
FFMA	FP32 Fused Multiply and Add
FMNMX	FP32 Minimum/Maximum

Opcode	Description
FMUL	FP32 Multiply
FSET	FP32 Compare And Set
FSETP	FP32 Compare And Set Predicate
FSWZADD	FP32 Add used for FSWZ emulation
MUFU	Multi Function Operation
RRO	Range Reduction Operator FP
DADD	FP64 Add
DFMA	FP64 Fused Mutiply Add
DMNMX	FP64 Minimum/Maximum
DMUL	FP64 Multiply
DSET	FP64 Compare And Set
DSETP	FP64 Compare And Set Predicate
HADD2	FP16 Add
HFMA2	FP16 Fused Mutiply Add
HMUL2	FP16 Multiply
HSET2	FP16 Compare And Set
HSETP2	FP16 Compare And Set Predicate
Integer Instructions	
BFE	Bit Field Extract
BFI	Bit Field Insert
FLO	Find Leading One
IADD	Integer Addition
IADD3	3-input Integer Addition
ICMP	Integer Compare to Zero and Select Source
IMAD	Integer Multiply And Add
IMADSP	Extracted Integer Multiply And Add.
IMNMX	Integer Minimum/Maximum
IMUL	Integer Multiply
ISCADD	Scaled Integer Addition
ISET	Integer Compare And Set
ISETP	Integer Compare And Set Predicate
LEA	Compute Effective Address
LOP	Logic Operation
LOP3	3-input Logic Operation
POPC	Population count
SHF	Funnel Shift
SHL	Shift Left
SHR	Shift Right

Opcode	Description
XMAD	Integer Short Multiply Add
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
Movement Instructions	
MOV	Move
PRMT	Permute Register Pair
SEL	Select Source with Predicate
SHFL	Warp Wide Register Shuffle
Predicate/CC Instructions	
CSET	Test Condition Code And Set
CSETP	Test Condition Code and Set Predicate
PSET	Combine Predicates and Set
PSETP	Combine Predicates and Set Predicate
P2R	Move Predicate Register To Register
R2P	Move Register To Predicate/CC Register
Texture Instructions	
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4
TXQ	Texture Query
TEXS	Texture Fetch with scalar/non-vec4 source/destinations
TLD4S	Texture Load 4 with scalar/non-vec4 source/destinations
TLDS	Texture Load with scalar/non-vec4 source/destinations
Compute Load/Store Instructions	
LD	Load from generic Memory
LDC	Load Constant
LDG	Load from Global Memory
LDL	Load within Local Memory Window
LDS	Local within Shared Memory Window
ST	Store to generic Memory
STG	Store to global Memory
STL	Store within Local or Shared Window
STS	Store within Local or Shared Window
ATOM	Atomic Operation on generic Memory
ATOMS	Atomic Operation on Shared Memory

Opcode	Description
RED	Reduction Operation on generic Memory
CCTL	Cache Control
CCTLL	Cache Control
MEMBAR	Memory Barrier
CCTLT	Texture Cache Control
Surface Memory Instructions	
SUATOM	Atomic Op on Surface Memory
SULD	Surface Load
SURED	Reduction Op on Surface Memory
SUST	Surface Store
Control Instructions	
BRA	Relative Branch
BRX	Relative Branch Indirect
JMP	Absolute Jump
JMX	Absolute Jump Indirect
SSY	Set Synchronization Point
SYNC	Converge threads after conditional branch
CAL	Relative Call
JCAL	Absolute Call
PRET	Pre-Return From Subroutine
RET	Return From Subroutine
BRK	Break
PBK	Pre-Break
CONT	Continue
PCNT	Pre-continue
EXIT	Exit Program
PEXIT	Pre-Exit
BPT	BreakPoint/Trap
Miscellaneous Instructions	
NOP	No Operation
CS2R	Move Special Register to Register
S2R	Move Special Register to Register
B2R	Move Barrier To Register
BAR	Barrier Synchronization
R2B	Move Register to Barrier
VOTE	Vote Across SIMD Thread Group

4.3. Volta Instruction Set

The Volta architecture (Compute Capability 7.x) has the following instruction set format:

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

Valid destination and source locations include:

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

Table 6 lists valid instructions for the Volta GPUs.

Table 6. Volta 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
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

Opcode	Description
DADD	FP64 Add
DFMA	FP64 Fused Mutiply Add
DMUL	FP64 Multiply
DSETP	FP64 Compare And Set Predicate
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
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

Opcode	Description
FRND	Round To Integer
Movement Instructions	
MOV	Move
MOV32I	Move
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	
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
ST	Store to Generic Memory
STG	Store to Global Memory
STL	Store within Local or Shared Window
STS	Store within Local or Shared Window
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
Texture Instructions	
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4

Opcode	Description
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
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
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
R2B	Move Register to Barrier
S2R	Move Special Register to Register

Opcode	Description
SETCTAID	Set CTA ID
SETLMEMBASE	Set Local Memory Base Address
VOTE	Vote Across SIMD Thread Group

4.4. Turing Instruction Set

The Turing architecture (Compute Capability 7.5) 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
- c[X][Y] for constant memory

Table 7 lists valid instructions for the Turing GPUs.

Table 7. 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
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

Opcode	Description
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
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

Opcode	Description
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
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 within Local or Shared Window
STS	Store within Local or Shared Window
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

Opcode	Description
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
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	
TEX	Texture Fetch
TLD	Texture Load
TLD4	Texture Load 4

Opcode	Description
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
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

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

4.5. Ampere Instruction Set

The Ampere architecture (Compute Capability 8.0) 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
- c[X][Y] for constant memory

Table 8 lists valid instructions for the Ampere GPUs.

Table 8. Ampere 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
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

Opcode	Description
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
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

Opcode	Description
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
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 within Local or Shared Window
STS	Store within Local or Shared Window
MATCH	Match Register Values Across Thread Group

Opcode	Description
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
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
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

Opcode	Description
USHR	Uniform Right Shift
VOTEU	Voting across SIMD Thread Group with Results in Uniform Destination
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	
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

Opcode	Description
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
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

Chapter 5. nvprune

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

5.1. 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_35 cubin rather than all the targets which normally exist:

```
nvprune -arch sm_35 libcublas_static.a -o libcublas_static35.a
```

Note that this means that libcublas_static35.a will not run on any other architecture, so should only be used when you are building for a single architecture.

5.2. Command-line Options

Table 9 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 9. `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.

Option (long)	Option (short)	Description
`--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'.
`--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.