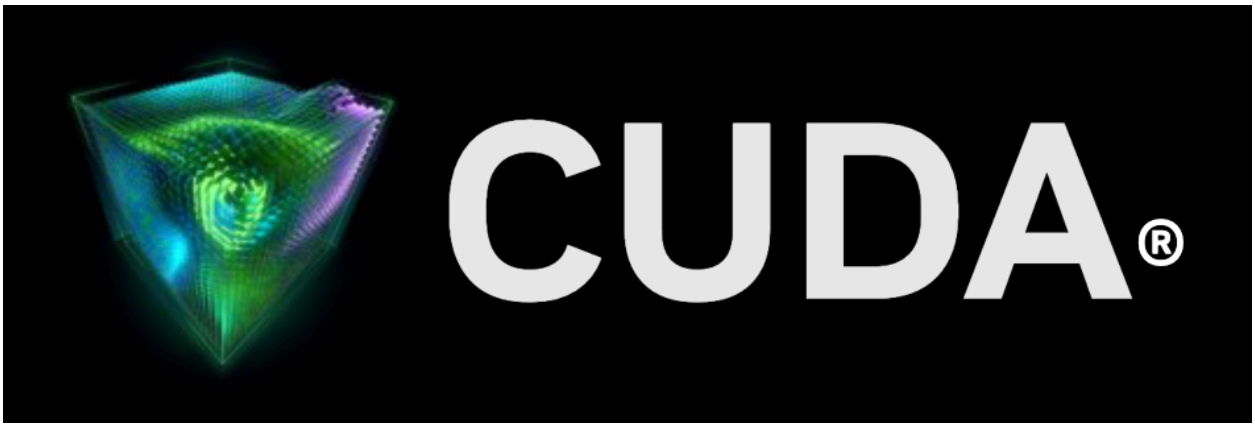


# nvFatbin

***Release 12.5***

**NVIDIA**

**May 09, 2024**

# Contents

## nvFatbin

The User guide to nvFatbin library.

The Fatbin Creator APIs are a set of APIs which can be used at runtime to combine multiple CUDA objects into one CUDA fat binary (fatbin).

The APIs accept inputs in multiple formats, either device cubins, PTX, or LTO-IR. The output is a fatbin that can be loaded by `cuModuleLoadData` of the CUDA Driver API.

The functionality in this library is similar to the `fatbinary` offline tool in the CUDA toolkit, with the following advantages:

- Support for runtime fatbin creation.
- The clients get fine grain control over the input process.
- Supports direct input from memory, rather than requiring inputs be written to files.

# Chapter 1. Getting Started

## 1.1. System Requirements

The Fatbin Creator library requires no special system configuration. It does not require a GPU.

## 1.2. Installation

The Fatbin Creator library is part of the CUDA Toolkit release and the components are organized as follows in the CUDA toolkit installation directory:

- On Windows:
  - `include\nvFatbin.h`
  - `lib\x64\nvFatbin.dll`
  - `lib\x64\nvFatbin_static.lib`
  - `doc\pdf\nvFatbin_User_Guide.pdf`
- On Linux:
  - `include/nvFatbin.h`
  - `lib64/libnvfatbin.so`
  - `lib64/libnvfatbin_static.a`
  - `doc/pdf/nvFatbin_User_Guide.pdf`

# Chapter 2. User Interface

This chapter presents the Fatbin Creator APIs. Basic usage of the API is explained in Basic Usage.

- Error codes
- Creation
- Supported Options

## 2.1. Error codes

**Enumerations**

***nvFatbinResult***
The enumerated type nvFatbinResult defines API call result codes.

**Functions**

**const char * *nvFatbinGetErrorString*(nvFatbinResult result)**
nvFatbinGetErrorString returns an error description string for each error code.

### 2.1.1. Enumerations

enum **nvFatbinResult**

The enumerated type nvFatbinResult defines API call result codes.

nvFatbin APIs return nvFatbinResult codes to indicate the result.

*Values:*

enumerator **NVFATBIN_SUCCESS**

enumerator **NVFATBIN_ERROR_INTERNAL**

enumerator **NVFATBIN_ERROR_ELF_ARCH_MISMATCH**

enumerator **NVFATBIN_ERROR_ELF_SIZE_MISMATCH**

enumerator **NVFATBIN_ERROR_MISSING_PTX_VERSION**

enumerator **NVFATBIN_ERROR_NULL_POINTER**

enumerator **NVFATBIN_ERROR_COMPRESSION_FAILED**

enumerator **NVFATBIN_ERROR_COMPRESSED_SIZE_EXCEEDED**

enumerator **NVFATBIN_ERROR_UNRECOGNIZED_OPTION**

enumerator **NVFATBIN_ERROR_INVALID_ARCH**

enumerator **NVFATBIN_ERROR_INVALID_NVVM**

enumerator **NVFATBIN_ERROR_EMPTY_INPUT**

enumerator **NVFATBIN_ERROR_MISSING_PTX_ARCH**

enumerator **NVFATBIN_ERROR_PTX_ARCH_MISMATCH**

enumerator **NVFATBIN_ERROR_MISSING_FATBIN**

enumerator **NVFATBIN_ERROR_INVALID_INDEX**

enumerator **NVFATBIN_ERROR_IDENTIFIER_REUSE**

## 2.1.2. Functions

const char ***nvFatbinGetErrorString**(*nvFatbinResult* result)

nvFatbinGetErrorString returns an error description string for each error code.

**Parameters**

**result** – **[in]** error code

**Returns**

- nullptr, if result is NVFATBIN_SUCCESS
- a string, if result is not NVFATBIN_SUCCESS

## 2.2. Fatbinary Creation

### Functions

**nvFatbinResult *nvFatbinAddCubin*(nvFatbinHandle handle, const void *code, size_t size, const char *arch, const char *identifier)**
nvFatbinAddCubin adds a CUDA binary to the fatbinary.

**nvFatbinResult *nvFatbinAddIndex*(nvFatbinHandle handle, const void *code, size_t size, const char *identifier)**
nvFatbinAddIndex adds an index file to the fatbinary.

**nvFatbinResult *nvFatbinAddLTOIR*(nvFatbinHandle handle, const void *code, size_t size, const char *arch, const char *identifier, const char *optionsCmdLine)**
nvFatbinAddLTOIR adds LTOIR to the fatbinary.

**nvFatbinResult *nvFatbinAddPTX*(nvFatbinHandle handle, const char *code, size_t size, const char *arch, const char *identifier, const char *optionsCmdLine)**
nvFatbinAddPTX adds PTX to the fatbinary.

**nvFatbinResult *nvFatbinAddReloc*(nvFatbinHandle handle, const void *code, size_t size)**
nvFatbinAddReloc adds relocatable PTX entries from a host object to the fatbinary.

**nvFatbinResult *nvFatbinCreate*(nvFatbinHandle *handle_indirect, const char **options, size_t optionsCount)**
nvFatbinCreate creates a new handle

**nvFatbinResult *nvFatbinDestroy*(nvFatbinHandle *handle_indirect)**
nvFatbinDestroy destroys the handle.

**nvFatbinResult *nvFatbinGet*(nvFatbinHandle handle, void *buffer)**
nvFatbinGet returns the completed fatbinary.

**nvFatbinResult *nvFatbinSize*(nvFatbinHandle handle, size_t *size)**
nvFatbinSize returns the fatbinary's size.

**nvFatbinResult *nvFatbinVersion*(unsigned int *major, unsigned int *minor)**
nvFatbinVersion returns the current version of nvFatbin

### Typedefs

***nvFatbinHandle***
nvFatbinHandle is the unit of fatbin creation, and an opaque handle for a program.

## 2.2.1. Functions

*nvFatbinResult* **nvFatbinAddCubin**(*nvFatbinHandle* handle, const void *code, size_t size, const char *arch, const char *identifier)

nvFatbinAddCubin adds a CUDA binary to the fatbinary.

User is responsible for making sure all strings are well-formed.

**Parameters**

- **handle** – **[in]** nvFatbin handle.
- **code** – **[in]** The cubin.
- **size** – **[in]** The size of the cubin.
- **arch** – **[in]** The architecture that this cubin is for.
- **identifier** – **[in]** Name of the cubin, useful when extracting the fatbin with tools like cuobjdump.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_INVALID_ARCH*
- *NVFATBIN_ERROR_ELF_ARCH_MISMATCH*
- *NVFATBIN_ERROR_ELF_SIZE_MISMATCH*
- *NVFATBIN_ERROR_COMPRESSION_FAILED,*
- *NVFATBIN_ERROR_UNRECOGNIZED_OPTION*
- *NVFATBIN_ERROR_COMPRESSED_SIZE_EXCEEDED*
- *NVFATBIN_ERROR_EMPTY_INPUT*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinAddIndex**(*nvFatbinHandle* handle, const void *code, size_t size, const char *identifier)

nvFatbinAddIndex adds an index file to the fatbinary.

User is responsible for making sure all strings are well-formed.

**Parameters**

- **handle** – **[in]** nvFatbin handle.
- **code** – **[in]** The index.
- **size** – **[in]** The size of the index.
- **identifier** – **[in]** Name of the index, useful when extracting the fatbin with tools like cuobjdump.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_INVALID_INDEX*
- *NVFATBIN_ERROR_COMPRESSION_FAILED,*
- *NVFATBIN_ERROR_UNRECOGNIZED_OPTION*
- *NVFATBIN_ERROR_COMPRESSED_SIZE_EXCEEDED*
- *NVFATBIN_ERROR_EMPTY_INPUT*

- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinAddLTOIR**(*nvFatbinHandle* handle, const void *code, size_t size, const char *arch, const char *identifier, const char *optionsCmdLine)

nvFatbinAddLTOIR adds LTOIR to the fatbinary.

User is responsible for making sure all strings are well-formed.

**Parameters**

- **handle** – **[in]** nvFatbin handle.
- **code** – **[in]** The LTOIR code.
- **size** – **[in]** The size of the LTOIR code.
- **arch** – **[in]** The architecture that this LTOIR is for.
- **identifier** – **[in]** Name of the LTOIR, useful when extracting the fatbin with tools like cuobjdump.
- **optionsCmdLine** – **[in]** Options used during JIT compilation.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_INVALID_ARCH*
- *NVFATBIN_ERROR_COMPRESSION_FAILED,*
- *NVFATBIN_ERROR_UNRECOGNIZED_OPTION*
- *NVFATBIN_ERROR_COMPRESSED_SIZE_EXCEEDED*
- *NVFATBIN_ERROR_EMPTY_INPUT*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinAddPTX**(*nvFatbinHandle* handle, const char *code, size_t size, const char *arch, const char *identifier, const char *optionsCmdLine)

nvFatbinAddPTX adds PTX to the fatbinary.

User is responsible for making sure all string are well-formed. The size should be inclusive of the terminating null character ('\0'). If the final character is not '\0', one will be added automatically, but in doing so, the code will be copied if it hasn't already been copied.

**Parameters**

- **handle** – **[in]** nvFatbin handle.
- **code** – **[in]** The PTX code.
- **size** – **[in]** The size of the PTX code.
- **arch** – **[in]** The architecture that this PTX is for.

- **identifier** – **[in]** Name of the PTX, useful when extracting the fatbin with tools like cuobjdump.
- **optionsCmdLine** – **[in]** Options used during JIT compilation.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_INVALID_ARCH*
- *NVFATBIN_ERROR_PTX_ARCH_MISMATCH*
- *NVFATBIN_ERROR_COMPRESSION_FAILED,*
- *NVFATBIN_ERROR_UNRECOGNIZED_OPTION*
- *NVFATBIN_ERROR_COMPRESSED_SIZE_EXCEEDED*
- *NVFATBIN_ERROR_EMPTY_INPUT*
- *NVFATBIN_ERROR_MISSING_PTX_VERSION*
- *NVFATBIN_ERROR_MISSING_PTX_ARCH*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinAddReloc**(*nvFatbinHandle* handle, const void *code, size_t size)

nvFatbinAddReloc adds relocatable PTX entries from a host object to the fatbinary.

Note that each relocatable ptx source must have a unique identifier (the identifiers are taken from the object's entries). This is enforced as only one entry per sm of each unique identifier. Note also that handle options are ignored for this operation. Instead, the host object's options are copied over from each of its entries.

**Parameters**

- **handle** – **[in]** nvFatbin handle.
- **code** – **[in]** The host object image.
- **size** – **[in]** The size of the host object image code.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_INVALID_ARCH*
- *NVFATBIN_ERROR_PTX_ARCH_MISMATCH*
- *NVFATBIN_ERROR_COMPRESSION_FAILED,*
- *NVFATBIN_ERROR_UNRECOGNIZED_OPTION*
- *NVFATBIN_ERROR_COMPRESSED_SIZE_EXCEEDED*
- *NVFATBIN_ERROR_EMPTY_INPUT*
- *NVFATBIN_ERROR_MISSING_PTX_VERSION*

- *NVFATBIN_ERROR_MISSING_PTX_ARCH*
- *NVFATBIN_ERROR_IDENTIFIER_REUSE*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinCreate**(*nvFatbinHandle* *handle_indirect, const char **options, size_t optionsCount)

nvFatbinCreate creates a new handle

**Parameters**

- **handle_indirect** – **[out]** Address of nvFatbin handle
- **options** – **[in]** An array of strings, each containing a single option.
- **optionsCount** – **[in]** Number of options.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_UNRECOGNIZED_OPTION*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinDestroy**(*nvFatbinHandle* *handle_indirect)

nvFatbinDestroy destroys the handle.

Use of any other pointers to the handle after calling this will result in undefined behavior. The passed in handle will be set to nullptr.

**Parameters**
**handle_indirect** – **[in]** Pointer to the handle.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinGet**(*nvFatbinHandle* handle, void *buffer)

nvFatbinGet returns the completed fatbinary.

User is responsible for making sure the buffer is appropriately sized for the `fatbinary`. You must call nvFatbinSize before using this, otherwise, it will return an error.

**See also:**

*nvFatbinSize*

**Parameters**

- **handle** – **[in]** nvFatbin handle.

- **buffer** – **[out]** memory to store fatbinary.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinSize**(*nvFatbinHandle* handle, size_t *size)

nvFatbinSize returns the fatbinary's size.

**Parameters**

- **handle** – **[in]** nvFatbin handle.
- **size** – **[out]** The fatbinary's size

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_INTERNAL*

*nvFatbinResult* **nvFatbinVersion**(unsigned int *major, unsigned int *minor)

nvFatbinVersion returns the current version of nvFatbin

**Parameters**

- **major** – **[out]** The major version.
- **minor** – **[out]** The minor version.

**Returns**

- *NVFATBIN_SUCCESS*
- *NVFATBIN_ERROR_NULL_POINTER*
- *NVFATBIN_ERROR_INTERNAL*

## 2.2.2. Typedefs

typedef struct _nvFatbinHandle ***nvFatbinHandle**

nvFatbinHandle is the unit of fatbin creation, and an opaque handle for a program.

To create a fatbin, an instance of nvFatbinHandle must be created first with *nvFatbinCreate()*.

## 2.3. Supported Options

nvFatbin supports the options below.

Option names are prefixed with a single dash (-). Options that take a value have an assignment operator (=) followed by the option value, with no spaces, e.g. "`-host=windows`".

The supported options are:

- `-32` Make entries 32 bit.
- `-64` Make entries 64 bit.
- `-c` Has no effect. (Deprecated, will be removed in the next major release. Didn't do anything from the start.)
- `-compress=<bool>` Enable (true) / disable (false) compression (default: true).
- `-compress-all` Compress everything in the fatbin, even if it's small.
- `-cuda` Specify CUDA (rather than OpenCL).
- `-g` Generate debug information.
- `-host=<name>` Specify host operating system. Valid options are "linux", "windows", and "mac" (deprecated).
- `-opencl` Specify OpenCL (rather than CUDA).

# Chapter 3. Basic Usage

This section of the document uses a simple example to explain how to use the Fatbin Creator APIs to link a program. For brevity and readability, error checks on the API return values are not shown.

This example assumes we want to create a fatbin with a CUBIN for sm_52, PTX for sm_61, and LTOIR for sm_70. We can create an instance of the fatbin creator and obtain an api handle to it as shown in Figure 1.

Figure 1. Fatbin Creator creation and initialization of a program

```
nvFatbinHandle handle;
nvFatbinCreate(&handle, nullptr, 0);
```

Assume that we already have three inputs stored in `std::vector` 's (CUBIN, PTX, and LTOIR), which could be from code created with `nvrtc` and stored into vectors. (They do not have to be in vectors, this merely illustrates that both the data itself and its size are needed.) We can add the inputs as shown in Figure 2.

Figure 2. Inputs to the fatbin creator

```
nvFatbinAddCubin(handle, cubin.data(), cubin.size(), "52", nullptr);
nvFatbinAddPTX(handle, ptx.data(), ptx.size(), "61", nullptr, nullptr);
nvFatbinAddLTOIR(handle, ltoir.data(), ltoir.size(), "70", nullptr, nullptr);
```

The fatbin can now be obtained. To obtain this we first allocate memory for it. And to allocate memory, we need to query the size of the fatbin which is done as shown in Figure 3.

Figure 3. Query size of the created fatbin

```
nvFatbinSize(linker, &fatbinSize);
```

The fatbin can now be queried as shown in Figure 4. This fatbin can then be executed on the GPU by passing this to the CUDA Driver APIs.

Figure 4. Query the created fatbin

```
void* fatbin = malloc(fatbinSize);
nvFatbinGet(handle, fatbin);
```

When the fatbin creator is not needed anymore, it can be destroyed as shown in Figure 5.

Figure 5. Destroy the fatbin creator

```
nvFatbinDestroy(&handle);
```

# Chapter 4. Compatibility

The nvFatbin library is compatible across releases. The library major version itself must be >= the maximum major version of the inputs.

For example, you can create a fatbin from a cubin created with 11.8 and one with 12.4 if your nvFatbin library is at least version 12.x.

# Chapter 5. Example: Runtime fatbin creation

This section demonstrates runtime fatbin creation. There are two cubins. The cubins are generated online using NVRTC.

These two cubins are then passed to `nvFatbin*` API functions, which put the cubins into a fatbin.

Note that this example requires a compatible GPU with drivers and NVRTC to work, even though the library doesn't require either.

## 5.1. Code (online.cpp)

```
#include <nvrtc.h>
#include <cuda.h>
#include <nvFatbin.h>
#include <nvrtc.h>
#include <iostream>

#define NUM_THREADS 128
#define NUM_BLOCKS 32

#define NVRTC_SAFE_CALL(x)                                        \
do {                                                              \
   nvrtcResult result = x;                                        \
   if (result != NVRTC_SUCCESS) {                                 \
      std::cerr << "\nerror: " #x " failed with error "           \
                << nvrtcGetErrorString(result) << '\n';           \
      exit(1);                                                    \
   }                                                              \
} while(0)

#define CUDA_SAFE_CALL(x)                                         \
do {                                                              \
   CUresult result = x;                                           \
   if (result != CUDA_SUCCESS) {                                  \
      const char *msg;                                            \
      cuGetErrorName(result, &msg);                               \
      std::cerr << "\nerror: " #x " failed with error "           \
                << msg << '\n';                                   \
      exit(1);                                                    \
```

(continues on next page)

(continued from previous page)

```
   }                                                                   \
} while(0)

#define NVFATBIN_SAFE_CALL(x)                                  \
do                                                             \
{                                                              \
   nvFatbinResult result = x;                                  \
   if (result != NVFATBIN_SUCCESS)                             \
   {                                                           \
      std::cerr << "\nerror: " #x " failed with error "   \
                << nvFatbinGetErrorString(result) << '\n';\
      exit(1);                                                 \
   }                                                           \
} while (0)

const char *fatbin_saxpy = "                                          \n\
__device__  float compute(float a, float x, float y) {        \n\
return a * x + y;                                             \n\
}                                                             \n\
                                                              \n\
extern \"C\" __global__                                       \n\
void saxpy(float a, float *x, float *y, float *out, size_t n) \n\
{                                                             \n\
size_t tid = blockIdx.x * blockDim.x + threadIdx.x;           \n\
if (tid < n) {                                                \n\
   out[tid] = compute(a, x[tid], y[tid]);                     \n\
}                                                             \n\
}                                                             \n";

size_t process(const void* input, const char* input_name, void** output, const char*
↪arch)
{
// Create an instance of nvrtcProgram with the code string.
nvrtcProgram prog;
NVRTC_SAFE_CALL(
nvrtcCreateProgram(&prog,         // prog
      (const char*) input,                       // buffer
      input_name,                 // name
      0,                          // numHeaders
      NULL,                       // headers
      NULL));                     // includeNames

const char *opts[1];
opts[0] = arch;
nvrtcResult compileResult = nvrtcCompileProgram(prog,  // prog
               1,     // numOptions
               opts); // options
// Obtain compilation log from the program.
size_t logSize;
NVRTC_SAFE_CALL(nvrtcGetProgramLogSize(prog, &logSize));
char *log = new char[logSize];
NVRTC_SAFE_CALL(nvrtcGetProgramLog(prog, log));
std::cout << log << '\n';
delete[] log;
if (compileResult != NVRTC_SUCCESS) {
exit(1);
```

(continues on next page)

(continued from previous page)

```
}
// Obtain generated CUBIN from the program.
size_t CUBINSize;
NVRTC_SAFE_CALL(nvrtcGetCUBINSize(prog, &CUBINSize));
char *CUBIN = new char[CUBINSize];
NVRTC_SAFE_CALL(nvrtcGetCUBIN(prog, CUBIN));
// Destroy the program.
NVRTC_SAFE_CALL(nvrtcDestroyProgram(&prog));
*output = (void*) CUBIN;
return CUBINSize;
}

int main(int argc, char *argv[])
{
void* known = NULL;
size_t known_size = process(fatbin_saxpy, "fatbin_saxpy.cu", &known, "-arch=sm_52");

CUdevice cuDevice;
CUcontext context;
CUmodule module;
CUfunction kernel;
CUDA_SAFE_CALL(cuInit(0));
CUDA_SAFE_CALL(cuDeviceGet(&cuDevice, 0));
CUDA_SAFE_CALL(cuCtxCreate(&context, 0, cuDevice));

// Dynamically determine the arch to make one of the entries of the fatbin with
int major = 0;
int minor = 0;
CUDA_SAFE_CALL(cuDeviceGetAttribute(&major,
                 CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MAJOR, cuDevice));
CUDA_SAFE_CALL(cuDeviceGetAttribute(&minor,
                 CU_DEVICE_ATTRIBUTE_COMPUTE_CAPABILITY_MINOR, cuDevice));
int arch = major*10 + minor;
char smbuf[16];
sprintf(smbuf, "-arch=sm_%d", arch);

void* dynamic = NULL;
size_t dynamic_size = process(fatbin_saxpy, "fatbin_saxpy.cu", &dynamic, smbuf);
sprintf(smbuf, "%d", arch);

// Load the dynamic CUBIN and the statically known arch CUBIN
// and put them in a fatbin together.
nvFatbinHandle handle;
const char* fatbin_options[] = {"-cuda"};
NVFATBIN_SAFE_CALL(nvFatbinCreate(&handle, fatbin_options, 1));

NVFATBIN_SAFE_CALL(nvFatbinAddCubin(handle,
                           (void *)dynamic, dynamic_size, smbuf, "dynamic"));
NVFATBIN_SAFE_CALL(nvFatbinAddCubin(handle,
                           (void *)known, known_size, "52", "known"));

size_t fatbinSize;
NVFATBIN_SAFE_CALL(nvFatbinSize(handle, &fatbinSize));
void *fatbin = malloc(fatbinSize);
NVFATBIN_SAFE_CALL(nvFatbinGet(handle, fatbin));
NVFATBIN_SAFE_CALL(nvFatbinDestroy(&handle));
```

(continues on next page)

(continued from previous page)

```
CUDA_SAFE_CALL(cuModuleLoadData(&module, fatbin));
CUDA_SAFE_CALL(cuModuleGetFunction(&kernel, module, "saxpy"));

// Generate input for execution, and create output buffers.
#define NUM_THREADS 128
#define NUM_BLOCKS 32
size_t n = NUM_THREADS * NUM_BLOCKS;
size_t bufferSize = n * sizeof(float);
float a = 5.1f;
float *hX = new float[n], *hY = new float[n], *hOut = new float[n];
for (size_t i = 0; i < n; ++i) {
   hX[i] = static_cast<float>(i);
   hY[i] = static_cast<float>(i * 2);
}
CUdeviceptr dX, dY, dOut;
CUDA_SAFE_CALL(cuMemAlloc(&dX, bufferSize));
CUDA_SAFE_CALL(cuMemAlloc(&dY, bufferSize));
CUDA_SAFE_CALL(cuMemAlloc(&dOut, bufferSize));
CUDA_SAFE_CALL(cuMemcpyHtoD(dX, hX, bufferSize));
CUDA_SAFE_CALL(cuMemcpyHtoD(dY, hY, bufferSize));
// Execute SAXPY.
void *args[] = { &a, &dX, &dY, &dOut, &n };
CUDA_SAFE_CALL(
   cuLaunchKernel(kernel,
                  NUM_BLOCKS, 1, 1,    // grid dim
                  NUM_THREADS, 1, 1,   // block dim
                  0, NULL,             // shared mem and stream
                  args, 0));           // arguments
CUDA_SAFE_CALL(cuCtxSynchronize());
// Retrieve and print output.
CUDA_SAFE_CALL(cuMemcpyDtoH(hOut, dOut, bufferSize));

for (size_t i = 0; i < n; ++i) {
   std::cout << a << " * " << hX[i] << " + " << hY[i]
             << " = " << hOut[i] << '\n';
}
// Release resources.
CUDA_SAFE_CALL(cuMemFree(dX));
CUDA_SAFE_CALL(cuMemFree(dY));
CUDA_SAFE_CALL(cuMemFree(dOut));
CUDA_SAFE_CALL(cuModuleUnload(module));
CUDA_SAFE_CALL(cuCtxDestroy(context));
delete[] hX;
delete[] hY;
delete[] hOut;
// Release resources.
free(fatbin);
delete[] ((char*)known);
delete[] ((char*)dynamic);

return 0;
}
```

## 5.2. Build Instructions

Assuming the environment variable CUDA_PATH points to CUDA Toolkit installation directory, build this example as:

- With nvFatbin shared library (note that if the test didn't use nvrtc or run the code then it would not need to link with nvrtc or the CUDA driver API):
  - Windows:

```
cl.exe online.cpp /Feonline ^
      /I "%CUDA_PATH%\include" ^
      "%CUDA_PATH%"\lib\x64\nvrtc.lib ^
      "%CUDA_PATH%"\lib\x64\nvfatbin.lib ^
      "%CUDA_PATH%"\lib\x64\cuda.lib
```

  - Linux:

```
g++ online.cpp -o online \
      -I $CUDA_PATH/include \
      -L $CUDA_PATH/lib64 \
      -lnvrtc -lnvfatbin -lcuda \
      -Wl,-rpath,$CUDA_PATH/lib64
```

- With nvFatbin static library:
  - Windows:

```
cl.exe online.cpp /Feonline  ^
      /I "%CUDA_PATH%"\include ^
      "%CUDA_PATH%"\lib\x64\nvrtc_static.lib ^
      "%CUDA_PATH%"\lib\x64\nvrtc-builtins_static.lib ^
      "%CUDA_PATH%"\lib\x64\nvfatbin_static.lib ^
      "%CUDA_PATH%"\lib\x64\nvptxcompiler_static.lib ^
      "%CUDA_PATH%"\lib\x64\cuda.lib user32.lib Ws2_32.lib
```

  - Linux:

```
g++ online.cpp -o online \
      -I $CUDA_PATH/include \
      -L $CUDA_PATH/lib64 \
      -lnvrtc_static -lnvrtc-builtins_static -lnvfatbin_static -
↪lnvptxcompiler_static -lcuda \
      -lpthread
```

## 5.3. Notices

### 5.3.1. 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

### 5.3.2. OpenCL

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

### 5.3.3. 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

# Index

## N