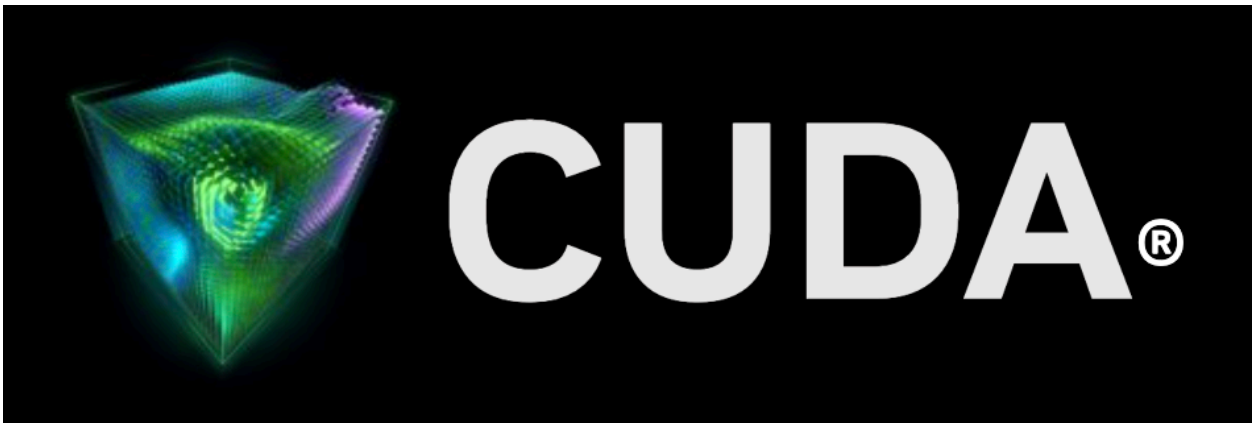

nvJitLink

Release 12.2

NVIDIA

Jun 22, 2023

Contents

nvJitLink

The User guide to nvJitLink library.

The JIT Link APIs are a set of APIs which can be used at runtime to link together GPU devide code.

The APIs accept inputs in multiple formats, either host objects, host libraries, fatbins, device cubins, PTX, or LTO-IR. The output is a linked cubin that can be loaded by `cuModuleLoadData` and `cuModuleLoadDataEx` of the CUDA Driver API.

Link Time Optimization can also be performed when given LTO-IR or higher level formats that include LTO-IR.

If an input does not contain GPU assembly code, it is first compiled and then linked.

The functionality in this library is similar to the `cuLink*` APIs in the CUDA Driver, with the following advantages:

- Support for Link Time Optimization
- Allow users to use runtime linking with the latest Toolkit version that is supported as part of CUDA Toolkit release. This support may not be available in the CUDA Driver APIs if the application is running with an older driver installed in the system. Refer to CUDA Compatibility for more details.
- The clients get fine grain control and can specify low-level compiler options during linking.

Chapter 1. Getting Started

1.1. System Requirements

The JIT Link library requires the following system configuration:

- POSIX threads support for non-Windows platform.
- GPU: Any GPU with CUDA Compute Capability 3.5 or higher.
- CUDA Toolkit and Driver.

1.2. Installation

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

- On Windows:
  - `include\nvJitLink.h`
  - `lib\x64\nvJitLink.dll`
  - `lib\x64\nvJitLink_static.lib`
  - `doc\pdf\nvJitLink_User_Guide.pdf`
- On Linux:
  - `include/nvJitLink.h`
  - `lib64/libnvJitLink.so`
  - `lib64/libnvJitLink_static.a`
  - `doc/pdf/nvJitLink_User_Guide.pdf`

Chapter 2. User Interface

This chapter presents the JIT Link APIs. Basic usage of the API is explained in Basic Usage.

- Error codes
- Linking
- Supported Link Options

2.1. Error codes

Enumerations

- *nvJitLinkResult*: The enumerated type nvJitLinkResult defines API call result codes.

2.1.1. Enumerations

enum **nvJitLinkResult**

The enumerated type nvJitLinkResult defines API call result codes.

nvJitLink APIs return nvJitLinkResult codes to indicate the result.

Values:

enumerator **NVJITLINK_SUCCESS**

enumerator **NVJITLINK_ERROR_UNRECOGNIZED_OPTION**

enumerator **NVJITLINK_ERROR_MISSING_ARCH**

enumerator **NVJITLINK_ERROR_INVALID_INPUT**

enumerator **NVJITLINK_ERROR_PTX_COMPILE**

enumerator **NVJITLINK_ERROR_NVVM_COMPILE**

enumerator **NVJITLINK_ERROR_INTERNAL**

2.2. Linking

Enumerations

- *nvJitLinkInputType*: The enumerated type nvJitLinkInputType defines the kind of inputs that can be passed to nvJitLinkAdd* APIs.

Functions

- *nvJitLinkAddData*: nvJitLinkAddData adds data image to the link.
- *nvJitLinkAddFile*: nvJitLinkAddFile reads data from file and links it in.
- *nvJitLinkComplete*: nvJitLinkComplete does the actual link.
- *nvJitLinkCreate*: nvJitLinkCreate creates an instance of nvJitLinkHandle with the given input options, and sets the output parameter `handle` .
- *nvJitLinkDestroy*: nvJitLinkDestroy frees the memory associated with the given handle and sets it to NULL.
- *nvJitLinkGetErrorLog*: nvJitLinkGetErrorLog puts any error messages in the log.
- *nvJitLinkGetErrorLogSize*: nvJitLinkGetErrorLogSize gets the size of the error log.
- *nvJitLinkGetInfoLog*: nvJitLinkGetInfoLog puts any info messages in the log.
- *nvJitLinkGetInfoLogSize*: nvJitLinkGetInfoLogSize gets the size of the info log.
- *nvJitLinkGetLinkedCubin*: nvJitLinkGetLinkedCubin gets the linked cubin.
- *nvJitLinkGetLinkedCubinSize*: nvJitLinkGetLinkedCubinSize gets the size of the linked cubin.
- *nvJitLinkGetLinkedPtx*: nvJitLinkGetLinkedPtx gets the linked ptx.
- *nvJitLinkGetLinkedPtxSize*: nvJitLinkGetLinkedPtxSize gets the size of the linked ptx.

Typedefs

- *nvJitLinkHandle*: nvJitLinkHandle is the unit of linking, and an opaque handle for a program.

2.2.1. Enumerations

enum **nvJitLinkInputType**

The enumerated type nvJitLinkInputType defines the kind of inputs that can be passed to nvJitLinkAdd* APIs.

Values:

enumerator **NVJITLINK_INPUT_NONE**

enumerator **NVJITLINK_INPUT_CUBIN**

enumerator **NVJITLINK_INPUT_PTX**

enumerator **NVJITLINK_INPUT_LTOIR**

enumerator **NVJITLINK_INPUT_FATBIN**

enumerator **NVJITLINK_INPUT_OBJECT**

enumerator **NVJITLINK_INPUT_LIBRARY**

2.2.2. Functions

static inline *nvJitLinkResult* **nvJitLinkAddData**(*nvJitLinkHandle* handle, *nvJitLinkInputType* inputType, const void *data, size_t size, const char *name)

nvJitLinkAddData adds data image to the link.

Parameters

- **handle** – **[in]** nvJitLink handle.
- **inputType** – **[in]** kind of input.
- **data** – **[in]** pointer to data image in memory.
- **name** – **[in]** name of input object.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkAddFile**(*nvJitLinkHandle* handle, *nvJitLinkInputType* inputType, const char *fileName)

nvJitLinkAddFile reads data from file and links it in.

Parameters

- **handle** – **[in]** nvJitLink handle.
- **inputType** – **[in]** kind of input.
- **fileName** – **[in]** name of file.

Returns

- *NVJITLINK_SUCCESS*

- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkComplete**(*nvJitLinkHandle* handle)

nvJitLinkComplete does the actual link.

Parameters **handle** – **[in]** nvJitLink handle.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkCreate**(*nvJitLinkHandle* *handle, uint32_t numOptions, const char **options)

nvJitLinkCreate creates an instance of nvJitLinkHandle with the given input options, and sets the output parameter `handle`.

It supports options listed in *Supported Link Options*.

See also:

nvJitLinkDestroy

Parameters

- **handle** – **[out]** Address of nvJitLink handle.
- **numOptions** – **[in]** Number of options passed.
- **options** – **[in]** Array of size `numOptions` of option strings.

Returns

- *NVJITLINK_SUCCESS*
- *NVJITLINK_ERROR_UNRECOGNIZED_OPTION*
- *NVJITLINK_ERROR_MISSING_ARCH*
- *NVJITLINK_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkDestroy**(*nvJitLinkHandle* *handle)

nvJitLinkDestroy frees the memory associated with the given handle and sets it to NULL.

See also:

nvJitLinkCreate

Parameters **handle** – **[in]** Address of nvJitLink handle.

Returns

- *NVJITLINK_SUCCESS*

- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetErrorLog**(*nvJitLinkHandle* handle, char *log)

nvJitLinkGetErrorLog puts any error messages in the log.

User is responsible for allocating enough space to hold the `log`.

See also:

nvJitLinkGetErrorLogSize

Parameters

- **handle** – **[in]** nvJitLink handle.
- **log** – **[out]** The error log.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetErrorLogSize**(*nvJitLinkHandle* handle, size_t *size)

nvJitLinkGetErrorLogSize gets the size of the error log.

See also:

nvJitLinkGetErrorLog

Parameters

- **handle** – **[in]** nvJitLink handle.
- **size** – **[out]** Size of the error log.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetInfoLog**(*nvJitLinkHandle* handle, char *log)

nvJitLinkGetInfoLog puts any info messages in the log.

User is responsible for allocating enough space to hold the `log`.

See also:

nvJitLinkGetInfoLogSize

Parameters

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

- **log** – **[out]** The info log.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetInfoLogSize**(*nvJitLinkHandle* handle, size_t *size)

nvJitLinkGetInfoLogSize gets the size of the info log.

See also:

nvJitLinkGetInfoLog

Parameters

- **handle** – **[in]** nvJitLink handle.
- **size** – **[out]** Size of the info log.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetLinkedCubin**(*nvJitLinkHandle* handle, void *cubin)

nvJitLinkGetLinkedCubin gets the linked cubin.

User is responsible for allocating enough space to hold the `cubin`.

See also:

nvJitLinkGetLinkedCubinSize

Parameters

- **handle** – **[in]** nvJitLink handle.
- **cubin** – **[out]** The linked cubin.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetLinkedCubinSize**(*nvJitLinkHandle* handle, size_t *size)

nvJitLinkGetLinkedCubinSize gets the size of the linked cubin.

See also:

nvJitLinkGetLinkedCubin

Parameters

- **handle** – **[in]** nvJitLink handle.
- **size** – **[out]** Size of the linked cubin.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetLinkedPtx**(*nvJitLinkHandle* handle, char *ptx)

nvJitLinkGetLinkedPtx gets the linked ptx.

Linked PTX is only available when using the `-lto` option. User is responsible for allocating enough space to hold the `ptx`.

See also:

nvJitLinkGetLinkedPtxSize

Parameters

- **handle** – **[in]** nvJitLink handle.
- **ptx** – **[out]** The linked PTX.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

static inline *nvJitLinkResult* **nvJitLinkGetLinkedPtxSize**(*nvJitLinkHandle* handle, size_t *size)

nvJitLinkGetLinkedPtxSize gets the size of the linked ptx.

Linked PTX is only available when using the `-lto` option.

See also:

nvJitLinkGetLinkedPtx

Parameters

- **handle** – **[in]** nvJitLink handle.
- **size** – **[out]** Size of the linked PTX.

Returns

- *NVJITLINK_SUCCESS*
- *NVRTC_ERROR_INVALID_INPUT*
- *NVRTC_ERROR_INTERNAL*

2.2.3. Typedefs

typedef struct nvJitLink ***nvJitLinkHandle**

nvJitLinkHandle is the unit of linking, and an opaque handle for a program.

To link inputs, an instance of nvJitLinkHandle must be created first with nvJitLinkCreate().

2.3. Supported Link Options

nvJitLink supports the link 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. "`-arch=sm_90`".

The supported options are:

- `-arch=sm_<N>` Pass SM architecture value. See nvcc for valid values of <N>. Can use compute_<N> value instead if only generating PTX. This is a required option.
- `-maxrregcount=<N>` Maximum register count.
- `-time` Print timing information to InfoLog.
- `-verbose` Print verbose messages to InfoLog.
- `-lto` Do link time optimization.
- `-ptx` Emit ptx after linking instead of cubin; only supported with `-lto`
- `-O<N>` Optimization level.
- `-g` Generate debug information.
- `-lineinfo` Generate line information.
- `-ftz=<n>` Flush to zero.
- `-prec-div=<n>` Precise divide.
- `-prec-sqrt=<n>` Precise square root.
- `-fma=<n>` Fast multiply add.
- `-kernels-used=<name>` Pass list of kernels that are used; any not in the list can be removed. This option can be specified multiple times.
- `-variables-used=<name>` Pass list of variables that are used; any not in the list can be removed. This option can be specified multiple times.
- `-optimize-unused-variables` Normally device code optimization is limited by not knowing what the host code references. With this option it can assume that if a variable is not referenced in device code then it can be removed.
- `-Xptxas=<opt>` Pass <opt> to ptxas. This option can be called multiple times.
- `-Xnvvm=<opt>` Pass <opt> to nvvm. This option can be called multiple times.

Chapter 3. Basic Usage

This section of the document uses a simple example to explain how to use the JIT Link 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 link for sm_80, but whatever arch is installed on the system should be used. We can create the linker and obtain a handle to it as shown in Figure 1.

Figure 1. Linker creation and initialization of a program

```
nvJitLink_t linker;
const char* link_options[] = { "-arch=sm_80" };
nvJitLinkCreate(&linker, 1, link_options);
```

Assume that we already have two relocatable input files (a.o and b.o), which could be created with the `nvcc -dc` command. We can add the input files as show in Figure 2.

Figure 2. Inputs to linker

```
nvJitLinkAddFile(linker, NVJITLINK_INPUT_OBJECT, "a.o");
nvJitLinkAddFile(linker, NVJITLINK_INPUT_OBJECT, "b.o");
```

Now the actual link can be done as shown in Figure 3.

Figure 3. Linking of the PTX program

```
nvJitLinkComplete(linker);
```

The linked GPU assembly code 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 image of the linked GPU assembly code which is done as shown in Figure 4.

Figure 4. Query size of the linked assembly image

```
nvJitLinkGetLinkedCubinSize(linker, &cubinSize);
```

The image of the linked GPU assembly code can now be queried as shown in Figure 5. This image can then be executed on the GPU by passing this image to the CUDA Driver APIs.

Figure 5. Query the linked assembly image

```
elf = (char*) malloc(cubinSize);
nvJitLinkGetLinkedCubin(linker, (void*)elf);
```

When the linker is not needed anymore, it can be destroyed as shown in Figure 6.

Figure 6. Destroy the linker

```
nvJitLinkDestroy(&linker);
```

Chapter 4. Compatibility

The nvJitLink library is compatible across minor versions in a release, but may not be compatible across major versions. The library version itself must be >= the maximum version of the inputs, and the shared library version must be >= the version that was linked with.

For example, you can link an object created with 12.0 and one with 12.1 if your nvJitLink library is version 12.x where x >= 1. If it was linked with 12.1, then you can replace and use the nvJitLink shared library with any version 12.x where x >= 1. On the flip side, you cannot use 12.0 to link 12.1 objects, nor use 12.0 nvJitLink library to run 12.1 code.

Linking across major versions (like 11.x with 12.x) works for ELF and PTX inputs, but does not work with LTOIR inputs. If using LTO, then compatibility is only guaranteed within a major release.

Chapter 5. Example: Device LTO (link time optimization)

This section demonstrates device link time optimization (LTO). There are two units of LTO IR. The first unit is generated offline using `nvcc`, by specifying the architecture as '`-arch lto_XX`' (see offline.cu). The generated LTO IR is packaged in a fatbinary.

The second unit is generated online using NVRTC, by specifying the flag '`-dlto`' (see online.cpp).

These two units are then passed to `libnvJitLink*` API functions, which link together the LTO IR, run the optimizer on the linked IR, and generate a cubin (see online.cpp). The cubin is then loaded on the GPU and executed.

5.1. Code (offline.cu)

```
__device__  float compute(float a, float x, float y) {
  return a * x + y;
}
```

5.2. Code (online.cpp)

```
#include <nvrtc.h>
#include <cuda.h>
#include <nvJitLink.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);                                                    \
```

(continues on next page)

(continued from previous page)

```
    }                                                                   \
  } 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);                                                          \
    }                                                                   \
  } while(0)
#define NVJITLINK_SAFE_CALL(h,x)                                        \
  do {                                                                  \
    nvJitLinkResult result = x;                                         \
    if (result != NVJITLINK_SUCCESS) {                                  \
      std::cerr << "\nerror: " #x " failed with error "                 \
                << result << '\n';                                      \
      size_t lsize;                                                     \
      result = nvJitLinkGetErrorLogSize(h, &lsize);                     \
      if (result == NVJITLINK_SUCCESS && lsize > 0) {                   \
        char *log = (char*)malloc(lsize);                               \
    result = nvJitLinkGetErrorLog(h, log);                              \
    if (result == NVJITLINK_SUCCESS) {                                  \
      std::cerr << "error: " << log << '\n';                            \
      free(log);                                                        \
    }                                                                   \
      }                                                                 \
      exit(1);                                                          \
    }                                                                   \
  } while(0)

const char *lto_saxpy = "                                               \n\
extern __device__ float compute(float a, float x, float y);             \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";
```

```
int main(int argc, char *argv[])
{
  size_t numBlocks = 32;
  size_t numThreads = 128;
  // Create an instance of nvrtcProgram with the code string.
  nvrtcProgram prog;
  NVRTC_SAFE_CALL(
    nvrtcCreateProgram(&prog,                       // prog
                       lto_saxpy,                   // buffer
                       "lto_saxpy.cu",              // name
                       0,                           // numHeaders
```

(continues on next page)

(continued from previous page)

```
                      NULL,                          // headers
                      NULL));                        // includeNames

  // specify that LTO IR should be generated for LTO operation
  const char *opts[] = {"-dlto",
                        "--relocatable-device-code=true"};
  nvrtcResult compileResult = nvrtcCompileProgram(prog,  // prog
                                                  2,     // 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);
  }
  // Obtain generated LTO IR from the program.
  size_t LTOIRSize;
  NVRTC_SAFE_CALL(nvrtcGetLTOIRSize(prog, &LTOIRSize));
  char *LTOIR = new char[LTOIRSize];
  NVRTC_SAFE_CALL(nvrtcGetLTOIR(prog, LTOIR));
  // Destroy the program.
  NVRTC_SAFE_CALL(nvrtcDestroyProgram(&prog));

  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));

  // Load the generated LTO IR and the LTO IR generated offline
  // and link them together.
  nvJitLinkHandle handle;
  // Dynamically determine the arch to link for
  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);
  const char *lopts[] = {"-lto", smbuf};
  NVJITLINK_SAFE_CALL(handle, nvJitLinkCreate(&handle, 2, lopts));

  // NOTE: assumes "offline.fatbin" is in the current directory
  // The fatbinary contains LTO IR generated offline using nvcc
  NVJITLINK_SAFE_CALL(handle, nvJitLinkAddFile(handle, NVJITLINK_INPUT_FATBIN,
                                "offline.fatbin"));
  NVJITLINK_SAFE_CALL(handle, nvJitLinkAddData(handle, NVJITLINK_INPUT_LTOIR,
```

(continues on next page)

(continued from previous page)

```
                              (void *)LTOIR, LTOIRSize, "lto_online"));

  // The call to nvJitLinkComplete causes linker to link together the two
  // LTO IR modules (offline and online), do optimization on the linked LTO IR,
  // and generate cubin from it.
  NVJITLINK_SAFE_CALL(handle, nvJitLinkComplete(handle));
  size_t cubinSize;
  NVJITLINK_SAFE_CALL(handle, nvJitLinkGetLinkedCubinSize(handle, &cubinSize));
  void *cubin = malloc(cubinSize);
  NVJITLINK_SAFE_CALL(handle, nvJitLinkGetLinkedCubin(handle, cubin));
  NVJITLINK_SAFE_CALL(handle, nvJitLinkDestroy(&handle));
  CUDA_SAFE_CALL(cuModuleLoadData(&module, cubin));
  CUDA_SAFE_CALL(cuModuleGetFunction(&kernel, module, "saxpy"));

  // Generate input for execution, and create output buffers.
  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));
  free(cubin);
  delete[] hX;
  delete[] hY;
  delete[] hOut;
  delete[] LTOIR;
```

(continues on next page)

(continued from previous page)

```
  return 0;
}
```

5.3. Build Instructions

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

- Compile offline.cu to fatbinary containing LTO IR (change `lto_52` to a different `lto_XX` architecture as appropriate).

```
nvcc -arch lto_52 -rdc=true -fatbin offline.cu
```

- With nvJitLink shared library (note that if test didn't use nvrtc then it would not need to link with nvrtc):
  - Windows:

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

  - Linux:

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

- With nvJitLink static library (when linking with static library then need to also link with nvptxcompiler_static, but with is implicitly included):
  - 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\nvJitLink_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 -lnvJitLink_static -
↪lnvptxcompiler_static -lcuda \
      -lpthread
```

5.4. Notices

5.4.1. 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.

5.4.2. OpenCL

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

5.4.3. 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