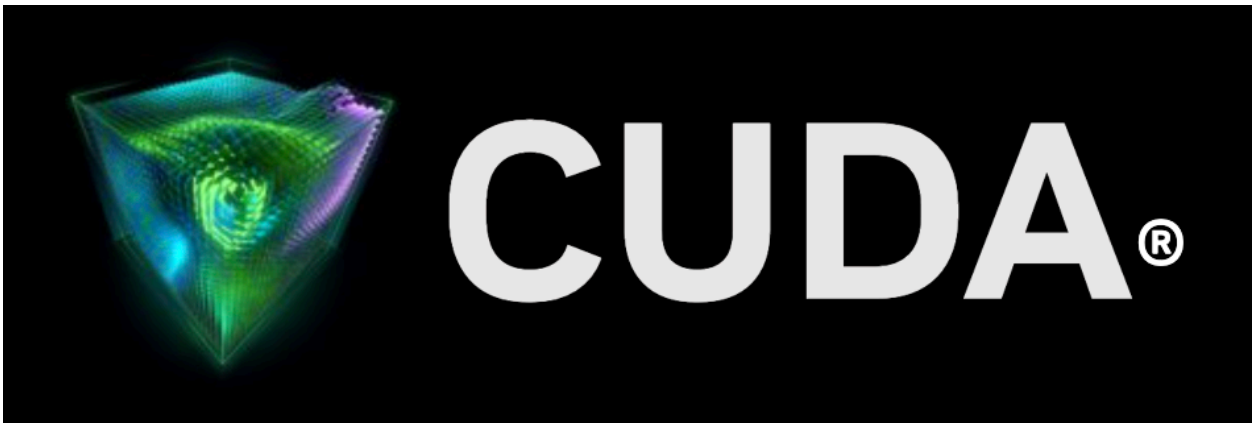

libNVVM API v4.0

Release 12.2

NVIDIA

Jul 18, 2023

Contents

libNVVM API

libNVVM API v4.0 Reference Manual

Chapter 1. Introduction

libNVVM API provides an interface for generating PTX code from both binary and text NVVM IR inputs. Compatible input can be generated by tools and libraries that produce LLVM 7.0 IR and bitcode. Support for reading the text NVVM IR representation is deprecated and may be removed in a later release.

Chapter 2. Thread Safety

Usage of libNVVM, via the libNVVM API, is intended to be thread-safe; however, users should expect serial execution performance.

Chapter 3. Module

This chapter presents the API of the libNVVM library. Here is a list of all modules:

- *Error Handling*
- *General Information Query*
- *Compilation*

Chapter 4. Error Handling

Enumerations

- *nvvmResult*: NVVM API call result code.

Functions

- *nvvmGetErrorString(nvvmResult result)*: Get the message string for the given *nvvmResult* code.

4.1. Enumerations

enum **nvvmResult**

NVVM API call result code.

Values:

enumerator **NVVM_SUCCESS**

enumerator **NVVM_ERROR_OUT_OF_MEMORY**

enumerator **NVVM_ERROR_PROGRAM_CREATION_FAILURE**

enumerator **NVVM_ERROR_IR_VERSION_MISMATCH**

enumerator **NVVM_ERROR_INVALID_INPUT**

enumerator **NVVM_ERROR_INVALID_PROGRAM**

enumerator **NVVM_ERROR_INVALID_IR**

enumerator **NVVM_ERROR_INVALID_OPTION**

enumerator **NVVM_ERROR_NO_MODULE_IN_PROGRAM**

enumerator **NVVM_ERROR_COMPILATION**

4.2. Functions

const char ***nvvmGetErrorString**(*nvvmResult* result)

Get the message string for the given *nvvmResult* code.

Parameters **result** – **[in]** NVVM API result code.

Returns Message string for the given *nvvmResult* code.

Chapter 5. General Information Query

Functions

- *nvvmIRVersion(int *majorIR, int *minorIR, int *majorDbg, int *minorDbg)*: Get the NVVM IR version.
- *nvvmVersion(int *major, int *minor)*: Get the NVVM version.

5.1. Functions

nvvmResult **nvvmIRVersion**(int *majorIR, int *minorIR, int *majorDbg, int *minorDbg)

Get the NVVM IR version.

Parameters

- **majorIR** – **[out]** NVVM IR major version number.
- **minorIR** – **[out]** NVVM IR minor version number.
- **majorDbg** – **[out]** NVVM IR debug metadata major version number.
- **minorDbg** – **[out]** NVVM IR debug metadata minor version number.

Returns

- *NVVM_SUCCESS*

nvvmResult **nvvmVersion**(int *major, int *minor)

Get the NVVM version.

Parameters

- **major** – **[out]** NVVM major version number.
- **minor** – **[out]** NVVM minor version number.

Returns

- *NVVM_SUCCESS*

Chapter 6. Compilation

Functions

- *nvvmAddModuleToProgram(nvvmProgram prog, const char *buffer, size_t size, const char *name)*: Add a module level NVVM IR to a program.
- *nvvmCompileProgram(nvvmProgram prog, int numOptions, const char **options)*: Compile the NVVM program.
- *nvvmCreateProgram(nvvmProgram *prog)*: Create a program, and set the value of its handle to `*prog`.
- *nvvmDestroyProgram(nvvmProgram *prog)*: Destroy a program.
- *nvvmGetCompiledResult(nvvmProgram prog, char *buffer)*: Get the compiled result.
- *nvvmGetCompiledResultSize(nvvmProgram prog, size_t *bufferSizeRet)*: Get the size of the compiled result.
- *nvvmGetProgramLog(nvvmProgram prog, char *buffer)*: Get the Compiler/Verifier Message.
- *nvvmGetProgramLogSize(nvvmProgram prog, size_t *bufferSizeRet)*: Get the Size of Compiler/Verifier Message.
- *nvvmLazyAddModuleToProgram(nvvmProgram prog, const char *buffer, size_t size, const char *name)*: Add a module level NVVM IR to a program.
- *nvvmVerifyProgram(nvvmProgram prog, int numOptions, const char **options)*: Verify the NVVM program.

Typedefs

- *nvvmProgram*: NVVM Program.

6.1. Functions

nvvmResult **`nvvmAddModuleToProgram`**(*nvvmProgram* prog, const char *buffer, size_t size, const char *name)

Add a module level NVVM IR to a program.

The `buffer` should contain an NVVM IR module. The module should have NVVM IR either in the LLVM 7.0.1 bitcode representation or in the LLVM 7.0.1 text representation. Support for reading the text representation of NVVM IR is deprecated and may be removed in a later version.

Parameters

- **prog** – **[in]** NVVM program.
- **buffer** – **[in]** NVVM IR module in the bitcode or text representation.
- **size** – **[in]** Size of the NVVM IR module.
- **name** – **[in]** Name of the NVVM IR module. If NULL, "<unnamed>" is used as the name.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_OUT_OF_MEMORY*
- *NVVM_ERROR_INVALID_INPUT*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmCompileProgram**(*nvvmProgram* prog, int numOptions, const char **options)

Compile the NVVM program.

The NVVM IR modules in the program will be linked at the IR level. The linked IR program is compiled to PTX.

The target datalayout in the linked IR program is used to determine the address size (32bit vs 64bit).

The valid compiler options are:

- -g (enable generation of full debugging information). Full debug support is only valid with '-opt=0'. Debug support requires the input module to utilize NVVM IR Debug Metadata. Line number (line info) only generation is also enabled via NVVM IR Debug Metadata, there is no specific libNVVM API flag for that case.
- -opt=
  - 0 (disable optimizations)
  - 3 (default, enable optimizations)
- -arch=
  - compute_50
  - compute_52 (default)
  - compute_53
  - compute_60
  - compute_61
  - compute_62
  - compute_70
  - compute_72
  - compute_75
  - compute_80
  - compute_87

- compute_89
- compute_90

- -ftz=
  - 0 (default, preserve denormal values, when performing single-precision floating-point operations)
  - 1 (flush denormal values to zero, when performing single-precision floating-point operations)
- -prec-sqrt=
  - 0 (use a faster approximation for single-precision floating-point square root)
  - 1 (default, use IEEE round-to-nearest mode for single-precision floating-point square root)
- -prec-div=
  - 0 (use a faster approximation for single-precision floating-point division and reciprocals)
  - 1 (default, use IEEE round-to-nearest mode for single-precision floating-point division and reciprocals)
- -fma=
  - 0 (disable FMA contraction)
  - 1 (default, enable FMA contraction)

Parameters

- **`prog`** – **[in]** NVVM program.
- **`numOptions`** – **[in]** Number of compiler `options` passed.
- **`options`** – **[in]** Compiler options in the form of C string array.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_OUT_OF_MEMORY*
- *NVVM_ERROR_IR_VERSION_MISMATCH*
- *NVVM_ERROR_INVALID_PROGRAM*
- *NVVM_ERROR_INVALID_OPTION*
- *NVVM_ERROR_NO_MODULE_IN_PROGRAM*
- *NVVM_ERROR_COMPILATION*

nvvmResult **`nvvmCreateProgram`**(*nvvmProgram* *prog)

Create a program, and set the value of its handle to `*prog`.

See also:

nvvmDestroyProgram()

Parameters **`prog`** – **[in]** NVVM program.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_OUT_OF_MEMORY*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmDestroyProgram**(*nvvmProgram* *prog)

Destroy a program.

See also:

nvvmCreateProgram()

Parameters `prog` – **[in]** NVVM program.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmGetCompiledResult**(*nvvmProgram* prog, char *buffer)

Get the compiled result.

The result is stored in the memory pointed to by `buffer`.

Parameters

- `prog` – **[in]** NVVM program.
- `buffer` – **[out]** Compiled result.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmGetCompiledResultSize**(*nvvmProgram* prog, size_t *bufferSizeRet)

Get the size of the compiled result.

Parameters

- `prog` – **[in]** NVVM program.
- `bufferSizeRet` – **[out]** Size of the compiled result (including the trailing NULL).

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmGetProgramLog**(*nvvmProgram* prog, char *buffer)

Get the Compiler/Verifier Message.

The NULL terminated message string is stored in the memory pointed to by `buffer` when the return value is NVVM_SUCCESS.

Parameters

- `prog` – **[in]** NVVM program.
- `buffer` – **[out]** Compilation/Verification log.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmGetProgramLogSize**(*nvvmProgram* prog, size_t *bufferSizeRet)

Get the Size of Compiler/Verifier Message.

The size of the message string (including the trailing NULL) is stored into `bufferSizeRet` when the return value is NVVM_SUCCESS.

Parameters

- **prog** – **[in]** NVVM program.
- **bufferSizeRet** – **[out]** Size of the compilation/verification log (including the trailing NULL).

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmLazyAddModuleToProgram**(*nvvmProgram* prog, const char *buffer, size_t size, const char *name)

Add a module level NVVM IR to a program.

The `buffer` should contain an NVVM IR module. The module should have NVVM IR in the LLVM 7.0.1 bitcode representation.

A module added using this API is lazily loaded - the only symbols loaded are those that are required by module(s) loaded using nvvmAddModuleToProgram. It is an error for a program to have all modules loaded using this API. Compiler may also optimize entities in this module by making them internal to the linked NVVM IR module, making them eligible for other optimizations. Due to these optimizations, this API to load a module is more efficient and should be used where possible.

Parameters

- **prog** – **[in]** NVVM program.
- **buffer** – **[in]** NVVM IR module in the bitcode representation.
- **size** – **[in]** Size of the NVVM IR module.
- **name** – **[in]** Name of the NVVM IR module. If NULL, "<unnamed>" is used as the name.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_OUT_OF_MEMORY*
- *NVVM_ERROR_INVALID_INPUT*
- *NVVM_ERROR_INVALID_PROGRAM*

nvvmResult **nvvmVerifyProgram**(*nvvmProgram* prog, int numOptions, const char **options)

Verify the NVVM program.

The valid compiler options are:

Same as for *nvvmCompileProgram()*.

See also:

nvvmCompileProgram()

Parameters

- **prog** – **[in]** NVVM program.
- **numOptions** – **[in]** Number of compiler `options` passed.
- **options** – **[in]** Compiler options in the form of C string array.

Returns

- *NVVM_SUCCESS*
- *NVVM_ERROR_OUT_OF_MEMORY*
- *NVVM_ERROR_IR_VERSION_MISMATCH*
- *NVVM_ERROR_INVALID_PROGRAM*
- *NVVM_ERROR_INVALID_IR*
- *NVVM_ERROR_INVALID_OPTION*
- *NVVM_ERROR_NO_MODULE_IN_PROGRAM*

6.2. Typedefs

typedef struct _nvvmProgram ***nvvmProgram**

NVVM Program.

An opaque handle for a program.

Chapter 7. Notices

7.1. 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.

7.2. OpenCL

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

7.3. 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