NVBLAS

Release 13.0

NVIDIA Corporation

Aug 01, 2025

Contents

1 NVBLAS Overview **3**

2 GPU Accelerated Routines **5**

3 BLAS Symbols Interception **7**

4 Device Memory Support **9**

5 Security Precaution **11**

6 Configuration **13**
6.1 NVBLAS_CONFIG_FILE Environment Variable . . . 13
6.2 Configuration Keywords . . . 13
6.2.1 NVBLAS_LOGFILE . . . 13
6.2.2 NVBLAS_TRACE_LOG_ENABLED . . . 14
6.2.3 NVBLAS_CPU_BLAS_LIB . . . 14
6.2.4 NVBLAS_GPU_LIST . . . 14
6.2.5 NVBLAS_TILE_DIM . . . 15
6.2.6 NVBLAS_GPU_DISABLED_<BLAS_FUNC_NAME> . . . 15
6.2.7 NVBLAS_CPU_RATIO_<BLAS_FUNC_NAME> . . . 15
6.2.8 NVBLAS_AUTOPIN_MEM_ENABLED . . . 15
6.2.9 Configuration File Example . . . 15

7 NVBLAS Installation **17**

8 Usage **19**

9 Notices **21**
9.1 Notice . . . 21
9.2 OpenCL . . . 22
9.3 Trademarks . . . 22

NVBLAS

The User guide for NVBLAS, drop-in BLAS replacement, multi-GPUs accelerated

The NVBLAS Library is a GPU-accelerated Libary that implements BLAS (Basic Linear Algebra Subprograms). It can accelerate most BLAS Level-3 routines by dynamically routing BLAS calls to one or more NVIDIA GPUs present in the system, when the charateristics of the call make it speed up on a GPU.

Chapter 1. NVBLAS Overview

The NVBLAS Library is built on top of the cuBLAS Library using only the CUBLASXT API (refer to the CUBLASXT API section of the cuBLAS Documentation for more details). NVBLAS also requires the presence of a CPU BLAS lirbary on the system. Currently NVBLAS intercepts only compute intensive BLAS Level-3 calls (see table below). Depending on the charateristics of those BLAS calls, NVBLAS will redirect the calls to the GPUs present in the system or to CPU. That decision is based on a simple heuristic that estimates if the BLAS call will execute for long enough to amortize the PCI transfers of the input and output data to the GPU. **Because NVBLAS does not support all standard BLAS routines, it might be necessary to associate it with an existing full BLAS Library. Please refer to the Usage section for more details.**

Chapter 2. GPU Accelerated Routines

NVBLAS offloads only the compute-intensive BLAS3 routines which have the best potential for acceleration on GPUs.

The following table shows the currently supported routines:

Routine	Types	Operation
gemm	S,D,C,Z	Multiplication of 2 matrices
syrk	S,D,C,Z	Symmetric rank-k update
herk	C,Z	Hermitian rank-k update
syr2k	S,D,C,Z	Symmetric rank-2k update
her2k	C,Z	Hermitian rank-2k update
trsm	S,D,C,Z	Triangular solve with multiple right-hand sides
trmm	S,D,C,Z	Triangular matrix-matrix multiplication
symm	S,D,C,Z	Symmetric matrix-matrix multiplication
hemm	C,Z	Hermitian matrix-matrix multiplication

Chapter 3. BLAS Symbols Interception

Standard BLAS Library implementations usually expose multiple symbols for the same routines. Let's say `func` is a BLAS routine name, `func_` or/and `func` are usually defined as extern symbols. Some BLAS Libraries might also expose some symbols with a proprietary appended prefix. NVBLAS intercepts only the symbols `func_` and `func`. The user needs to make sure that the application intended to be GPU-accelerated by NVBLAS actually calls those defined symbols. Any other symbols will not be intercepted and the original BLAS routine will be executed for those cases.

Chapter 4. Device Memory Support

Starting with Release 8.0, data can be located on any GPU device, even on GPU devices that are not configured to be part of the computation. When any of the data is located on a GPU, the computation will be exclusively done on GPU whatever the size of the problem. Also, this feature has to be used with caution: the user has to be sure that the BLAS call will indeed be intercepted by NVBLAS, otherwise it will result in a crash when the CPU BLAS tries to execute it.

Chapter 5. Security Precaution

Because the NVBLAS Library relies on a symbols interception mechanism, it is essential to make sure it has not been compromised. In that regard, NVBLAS should never be used from a process running at elevated privileges, such as Administrator on Windows or root on Linux.

Chapter 6. Configuration

Because NVBLAS is a drop-in replacement of BLAS, it must be configured through an ASCII text file that describes how many and which GPUs can participate in the intercepted BLAS calls. The configuration file is parsed at the time of the loading of the library. The format of the configuration file is based on keywords optionally followed by one or more user-defined parameters. At most one keyword per line is allowed. Blank lines or lines beginning with the character # are ignored.

6.1. NVBLAS_CONFIG_FILE Environment Variable

The location and name of the configuration file must be defined by the environment variable `NVBLAS_CONFIG_FILE`. By default, if `NVBLAS_CONFIG_FILE` is not defined, NVBLAS will try to open the file `nvblas.conf` in the current directory. For a safe use of NVBLAS, the configuration file should have have restricted write permissions.

6.2. Configuration Keywords

The configuration keywords syntax is described in the following subsections.

6.2.1. NVBLAS_LOGFILE

This keyword defines the file where NVBLAS should print status and error messages. By default, if not defined, the standard error output file (eg. stderr) will be used. It is advised to define this keyword early in the configuration to capture errors in parsing that file itself.

6.2.2. NVBLAS_TRACE_LOG_ENABLED

When this keyword is defined, every intercepted BLAS calls will be logged into the NVBLAS_LOGFILE. This feature, even though intrusive, can be useful for debugging purposes.

6.2.3. NVBLAS_CPU_BLAS_LIB

This keyword defines the CPU BLAS dynamic library file (for example, `.so` file on Linux or `.dll` on Windows) that NVBLAS should open to find the CPU BLAS symbols definitions. This keyword must be defined for NVBLAS to work. Because CPU Blas libraries are often composed of multiple files, even though this keyword is set to the full path to the main file of the CPU library, it might still be necessary to define the right path to find the rest of the library files in the environment of your system. On Linux, this can be done by setting the environment variable `LD_LIBRARY_PATH` whereas on Windows, this can be done by setting the environment variable `PATH`.

For a safe use of NVBLAS, the following precautions are strongly advised:

- The CPU BLAS Library should be located where ordinary users do not have write permissions.
- The path specified should be absolute, not relative.

6.2.4. NVBLAS_GPU_LIST

This keyword defines the list of GPUs that should participate in the computation of the intercepted BLAS calls. If not defined, only GPU device 0 is used, since that is normally the most compute-capable GPU installed in the system. This keyword can be set to a list of device numbers separated by blank characters. Also the following wildcard keywords are also accepted for simplicity :

Key-word	Meaning
`ALL`	All compute-capable GPUs detected on the system will be used by NVBLAS
`ALL0`	GPU device 0, AND all others GPUs detected that have the same compute-capabilities as device 0 will be used by NVBLAS

Note: In the current release of CUBLAS, the CUBLASXT API supports two GPUs if they are on the same board such as Tesla K10 or GeForce GTX690 and one GPU otherwise. Because NVBLAS is built on top of the CUBLASXT API, NVBLAS has the same restriction. If access to more GPUs devices is needed, details of the licensing are described at cublasXt.

6.2.5. NVBLAS_TILE_DIM

This keyword defines the tile dimension that should be used to divide the matrices involved in the computation. This definition maps directly to a call of the cublasXt API routine `cublasXtSetBlockDim`. Refer to cuBLAS documentation to understand the tradeoffs associated with setting this to a larger or a smaller value.

6.2.6. NVBLAS_GPU_DISABLED_<BLAS_FUNC_NAME>

This keyword, appended with the name of a BLAS routine disables NVBLAS from running a specified routine on the GPU. This feature is intended mainly for debugging purposes. By default, all supported BLAS routines are enabled.

6.2.7. NVBLAS_CPU_RATIO_<BLAS_FUNC_NAME>

This keyword, appended with the name of ta BLAS routine defines the ratio of the workload that should remain on the CPU in the event that the NVBLAS decides to offload work for that routine on the GPU. This functionality is directly mapped to the cublasXt API routine `cublasXtSetCpuRatio`. By default, the ratio is defined to zero for all routines. Please refer to the cuBLAS documentation for details and for the list of routines which support this feature.

6.2.8. NVBLAS_AUTOPIN_MEM_ENABLED

This keyword enables the Pinning Memory mode. This functionality is directly mapped to the cublasXt API routine `cublasXtSetPinningMemMode`. If this keyowrd is not present in the configuration file, the Pinning Memory mode will be set to `CUBLASXT_PINNING_DISABLED`.

Note: There are some restrictions to use this feature as specified in the cuBLAS documentation of the underlying routine `cublasXtSetPinningMemMode`. Specifically when NVBLAS is used in a multi-threaded applications, this option should not be used if there is a chance that matrices used by different threads overlaps while calling NVBLAS. Please refer to the cuBLAS Documentation of the routine `cublasXtSetPinningMemMode <https://docs.nvidia.com/cuda/cublas/index.html#cublasxt_setPinningMemMode>`__ for details.

6.2.9. Configuration File Example

The following example shows a typical NVBLAS configuration file :

```
# This is the configuration file to use NVBLAS Library
# Setup the environment variable NVBLAS_CONFIG_FILE to specify your own config file.
# By default, if NVBLAS_CONFIG_FILE is not defined,
# NVBLAS Library will try to open the file "nvblas.conf" in its current directory
# Example : NVBLAS_CONFIG_FILE  /home/cuda_user/my_nvblas.conf
```

(continues on next page)

(continued from previous page)

```
# The config file should have restricted write permissions accesses

# Specify which output log file (default is stderr)
NVBLAS_LOGFILE  nvblas.log

# Enable trace log of every intercepted BLAS calls
NVBLAS_TRACE_LOG_ENABLED

#Put here the CPU BLAS fallback Library of your choice
#It is strongly advised to use full path to describe the location of the CPU Library
NVBLAS_CPU_BLAS_LIB  /usr/lib/libopenblas.so
#NVBLAS_CPU_BLAS_LIB  <mkl_path_installtion>/libmkl_rt.so

# List of GPU devices Id to participate to the computation
# Use ALL if you want all your GPUs to contribute
# Use ALL0, if you want all your GPUs of the same type as device 0 to contribute
# However, NVBLAS consider that all GPU have the same performance and PCI bandwidth
# By default if no GPU are listed, only device 0 will be used

#NVBLAS_GPU_LIST 0 2 4
#NVBLAS_GPU_LIST ALL
NVBLAS_GPU_LIST ALL0

# Tile Dimension
NVBLAS_TILE_DIM 2048

# Autopin Memory
NVBLAS_AUTOPIN_MEM_ENABLED

#List of BLAS routines that are prevented from running on GPU (use for debugging
↪purpose
# The current list of BLAS routines supported by NVBLAS are
# GEMM, SYRK, HERK, TRSM, TRMM, SYMM, HEMM, SYR2K, HER2K

#NVBLAS_GPU_DISABLED_SGEMM
#NVBLAS_GPU_DISABLED_DGEMM
#NVBLAS_GPU_DISABLED_CGEMM
#NVBLAS_GPU_DISABLED_ZGEMM

# Computation can be optionally hybridized between CPU and GPU
# By default, GPU-supported BLAS routines are ran fully on GPU
# The option NVBLAS_CPU_RATIO_<BLAS_ROUTINE> give the ratio [0,1]
# of the amount of computation that should be done on CPU
# CAUTION : this option should be used wisely because it can actually
# significantly reduced the overall performance if too much work is given to CPU

#NVBLAS_CPU_RATIO_CGEMM 0.07
```

Chapter 7. NVBLAS Installation

The NVBLAS Library is part of the CUDA Toolkit, and will be installed along all the other CUDA libraries. It is available on 64-bit operating systems. NVBLAS Library is built on top of cuBLAS, so the cuBLAS library needs to be accessible by NVBLAS.

Chapter 8. Usage

To use the NVBLAS Library, the user application must be relinked against NVBLAS in addition to the original CPU Blas (technically only NVBLAS is needed unless some BLAS routines not supported by NVBLAS are used by the application). To be sure that the linker links against the exposed symbols of NVBLAS and not the ones from the CPU BLAS, the NVBLAS Library needs to be put before the CPU BLAS on the linkage command line.

On Linux, an alternative way to use NVBLAS Library is to use the `LD_PRELOAD` environment variable; this technique has the advantage of avoiding the relinkage step. However, the user should avoid defining that environment variable globally because it will cause the NVBLAS library to be loaded by every shell command executed on the system, thus leading to a lack of responsiveness of the system.

Finally, mathematical tools and libraries often offer the opportunity to specify the BLAS Library to be used through an environment variable or a configuration file. Because NVBLAS does not support all the standard BLAS routines, it might be necessary to pair NVBLAS with a full BLAS library, even though your application only calls supported NVBLAS routines. Fortunately, those tools and libraries usually offer a way to specify multiple BLAS Libraries. Please refer to the documentation of the appropriate tools and libraries for details.

Chapter 9. Notices

9.1. 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.

9.2. OpenCL

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

9.3. 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

©2007-2025, NVIDIA Corporation & affiliates. All rights reserved