NVIDIA NVSHMEM

Installation Guide

DG-08911-001_v2.9.0 | March 2023

Table of Contents

Chapter 1. Overview

NVIDIA® NVSHMEM™ is a programming interface that implements a Partitioned Global Address Space (PGAS) model across a cluster of NVIDIA GPUs. NVSHMEM provides an easy-to-use interface to allocate memory that is symmetrically distributed across the GPUs. In addition to a CPU-side interface, NVSHMEM also provides a CUDA kernel-side interface that allows NVIDIA CUDA® threads to access any location in the symmetrically-distributed memory.

Chapter 2. Hardware And Software Requirements

NVSHMEM has the following hardware and software requirements.

2.1. Hardware Requirements

NVSHMEM requires the following hardware:

- The `x86_64` or `ppc64le` CPU architectures.
- NVIDIA Data Center GPU of the NVIDIA Volta™ GPU architecture or later.

  For a complete list, refer to https://developer.nvidia.com/cuda-gpus.
- All GPUs must be P2P-connected via NVLink/PCIe or via GPUDirect RDMA over InfiniBand/RoCE with a Mellanox adapter (CX-4 or later).

  Support for atomics requires a NVLink connection or a GPUDirect RDMA connection and GDRCopy. Refer to Software Requirements for more information.

2.2. Software Requirements

NVSHMEM requires the following software:

- 64-bit Linux.

  For a complete compatibility matrix, see the NVIDIA CUDA Installation Guide for Linux .
- A C++ Compiler with C++11 support.
- CUDA 10.2 or later.
- GNU Make 3.81 or later.
- (Optional) InfiniBand GPUDirect Async (IBGDA) transport
  - Requires Mellanox OFED >= 5.0
  - Requires nvidia.ko >= 510.40.3 loaded with PeerMappingOverride=1. This can be accomplished by modifying the options in /etc/modprobe.d/nvidia.conf as follows: options nvidia NVreg_RegistryDwords="PeerMappingOverride=1;"
  - Requires nvidia_peermem >= 510.40.3 OR nv_peer_mem >= 1.3

NOTE: For more information, see: GPUDirect Async

- (Optional)Mellanox OFED.
  - This software is required to build the IBRC transport. If the OFED is unavailable, NVSHMEM can be built with `NVSHMEM_IBRC_SUPPORT=0` set in the environment.
- (Optional) `nv_peer_mem` for GPUDirect RDMA.
  - This software must use the IBRC and UCX transports and is required when `NVSHMEM_IBRC_SUPPORT=0` and `NVSHMEM_UCX_SUPPORT=0` are not set at compile time.

> Note: Both the IBRC and UCX transports make use of GDRCopy in order to perform atomic operations. If the user is using either of these transports and intend on performing atomic operations, they MUST enable GDRCopy support. All other transports do not depend on GDRCopy and it is not needed in those cases.

  - A PMI-1 (for example, Hydra), PMI-2 (for example, Slurm), or a PMIx (for example, Open MPI) compatible launcher.
- (Optional) GDRCopy v2.0 or later.
  - This software is required for atomics support on non-NVLink connections.
  - It is required when `NVSHMEM_IBRC-SUPPORT=0` and `NVSHMEM_UCX_SUPPORT=0` are not set at compile time.
- (Optional) UCX version 1.10.0 or later.
  - This software is required to build the UCX transport.

> Note: UCX must be configured with `--enable-mt` and `--with-dm`.

- (Optional) libfabric 1.15.0.0 or later
- (Optional) NCCL 2.0 or later.
- (Optional) PMIx 3.1.5 or later.
- (Optional) CMAKE 3.19 or later.

2.3. System Requirements

Here is some information about an additional system requirement.

(Optional) CUDA MPS Service

When using multiple processes-per-GPU, to support the complete NVHSMEM API, the CUDA MPS server must be configured on the system. To avoid deadlock situations, the total GPU Utilization that is shared between the processes must be capped at 100% or lower.

Refer to Multi-Process Service for more information about how to configure the MPS server.

Chapter 3. Installation

3.1. Downloading NVSHMEM

Procedure

Download and extract the NVSHMEM txz archive from https://developer.download.nvidia.com/compute/redist/nvshmem/version-number/source (for example, https://developer.download.nvidia.com/compute/redist/nvshmem/2.9.0/source/).

The extracted directory contains the following files and subdirectories:

File or Directory	Description
`src/`	Contains NVSHMEM sources and headers.
`perftest/`	Contains tests showing use of NVSHMEM APIs with performance reporting.
`examples/`	Contains examples showing use of some common use cases of NVSHMEM.
`scripts/`	Contains helper scripts, for example, the script to download, build, and install Hydra.
`changelog`	Change history for the repository.
`COPYRIGHT.txt`	Copyright information.
`NVSHMEM-SLA.txt`	NVSHMEM Software License Agreement (SLA).

3.2. Building And Installing NVSHMEM

Procedure

1. Set the `CUDA_HOME` environment variable to point to the CUDA Toolkit.
2. Set the `GDRCOPY_HOME` environment variable to point to the GDRCopy installation.

   To build without GDRCopy, set the environmental variable to `NVSHMEM_USE_GDRCOPY=0`.

Note: Without GDRCopy, atomics are only supported across NVLink connections.

3. If MPI and/or SHMEM support is required, set `NVSHMEM_MPI_SUPPORT=1` and/or `NVSHMEM_SHMEM_SUPPORT=1`.
4. Set the `MPI_HOME` and `SHMEM_HOME` environment variables to point to the MPI and OpenSHMEM installations, respectively.
5. By default, the location of `mpicc` that is used during NVSHMEM compilation is set to `$MPI_HOME/bin/mpicc`.

   This location can be overridden by specifying `MPICC=<path/to/mpicc>` in the environment.

   Note: Here is some additional information:

   - When using Open MPI and OSHMEM, the paths are the same.
   - To use OSHMEM, Open MPI needs to be built with UCX support.
   - NVSHMEM has been tested with Open MPI 4.0.1 and UCX 1.10.
   - Other MPI and OpenSHMEM installations should work.
   - By default, MPI support is enabled, and OpenSHMEM support is disabled.

6. Optional: To enable UCX support, set `NVSHMEM_UCX_SUPPORT=1` and `UCX_HOME` to the installed UCX directory.
7. Optional: The enable libfabric support, set `NVSHMEM_LIBFABRIC_SUPPORT=1` and `LIBFABRIC_HOME` to the installed libfabric directory.
8. Optional: To enable NCCL support, set `NVSHMEM_USE_NCCL=1` and `NCCL_HOME` to the installed NCCL directory.
9. Optional: To enable the InfiniBand GPUDirect Async (IBGDA) transport, set NVSHMEM_IBGDA_SUPPORT=1
10. Optional: To enable PMIx support, set `NVSHMEM_PMIX_SUPPORT=1` and`PMIX_HOME` to the installed PMIx directory.
11. Optional: Configure the default bootstrap:

    - The PMI bootstrap method can be selected by using the `NVSHMEM_BOOTSTRAP_PMI` environment variable at runtime.

      PMI-1, which can be used with the Hydra launcher, is the default PMI standard that is used by NVSHMEM.
    - To select PMIx as the default PMI interface, set `NVSHMEM_DEFAULT_PMIX=1`.
    - To select PMI-2 as the default PMI interface, set `NVSHMEM_DEFAULT_PMI2=1`.

12. (Optional) To enable all of device API implementation to be inlined, set `NVSHMEM_ENABLE_ALL_DEVICE_INLINING=1`. By default, this build variable is set to 0 and only P2P memory copying code is inlined while remote transfers and collectives API are not inlined. Setting this build variable to 1 enables inlining for all device API implementation by using __forceinline__ function qualifier.
13. Set `NVSHMEM_PREFIX` to specify the location where NVSHMEM will be installed.
14. To build and install the library, you can use cmake (preferred) or make directly (deprecated)

    - `cmake -S {sourcedir} -B {builddir} && make -C {builddir} -j install`
    - `Make -j8 install`

3.3. Using NVSHMEM In Your Applications

3.3.1. Using the NVSHMEM cmake build system

NVSHMEM now supports building with cmake version 3.19 or later.

The cmake build system is backwards compatible with the environment variables used in the original Makefile. That is to say that the same environment will produce a comparable build whether make or cmake is used for the build.

Cmake natively supports some environment and cmake variables for facilitating discovery of NVSHMEM dependencies (e.g. MPI and CUDA). These native settings can be used within the context of NVSHMEM, but describing them is outside the scope of this document.

Additionally, with the exception of NVSHMEM_HOME (which was superseded by NVSHMEM_PREFIX for the install prefix) all previous environment variables are respected when passed as cmake variables.

The steps outlined below describe typical build steps for NVSHMEM when using cmake:

1. If setting build configurations through the environment, follow steps 1-12 as listed in section 3.2 above.
2. To create the makefiles for building NVSHMEM call:
   ```
   cmake [-D{VAR_NAME_1={VAR_VAL_1}} ...-D{VAR_NAME_N={VAR_VAL_N}} -S . -B
    {PATH_TO_BUILD_DIR}
   ```
3. Where VAR_NAME_X can be any previously accepted environment variable or native cmake variable. Note: If the variables are already set in the environment, it is not necessary to set them again on the command line. If variables are set on both the command line and the environment, the command line will supersede the environment. To make NVSHMEM:
   ```
   cd {PATH_TO_BUILD_DIR} && make [-j] [install]'
   ```

   

   Note: CMAKE does not respect NVCC_GENCODE. Instead, use the cmake variable CUDA_ARCHITECTURES as follows: cmake -DCUDA_ARCHITECTURES="60;70”
4. When building with CMAKE from the source packages, perftests and examples can be disabled with the CMAKE by setting variables NVSHMEM_BUILD_TESTS and NVSHMEM_BUILD_EXAMPLES, respectively, to 0
5. Binary packages can be built from the source package by setting the CMake variable NVSHMEM_BUILD_PACKAGES to 1

   a). Users also have granular control over whether to produce RPM or DEB files with the CMake variables NVSHMEM_BUILD_DEB_PACKAGE and NVSHMEM_BUILD_RPM_PACKAGE. These are set to 1 by default.

3.3.2. Launching NVSHMEM Programs

- Using a PMI-1 compatible launcher, such as Hydra.
- Using a PMI-2 compatible launcher, such as Slurm.
- Using a PMIx compatible launcher, such as Slurm or Open MPI mpirun.
- Launching as part of an existing MPI application.
- Launching as part of an existing OpenSHMEM application.

The PMI-1 and PMI-2 clients are in NVSHMEM and are automatically built as part of the build process. A PMIx client must be provided by the user by installing Open PMIx or by using the PMIx client that is installed by Open MPI or Slurm. When you build Open MPI, include the `--enable-install-libpmix configure` option. When you build NVSHMEM, set `NVSHMEM_PMIX_SUPPORT=1` and `PMIX_HOME=/path/to/openmpi`.

To select the correct PMI library at runtime, set `NVSHMEM_BOOTSTRAP_PMI` to `PMI`, `PMI-2`, or `PMIx`. To bootstrap NVSHMEM by using MPI or OpenSHMEM, start the application in the typical way, start MPI or OpenSHMEM, and then call the `nvshmemx_init_attr` function to inform NVSHMEM that NVSHMEM is running as part of an existing MPI or OpenSHMEM job.

3.3.3. Using NVSHMEM with Multiple Processes-Per-GPU

Starting with release 2.5.0, NVSHMEM supports multiple processes-per-GPU (MPG), which does not require additional configuration and can be run with or without the CUDA Multi-process Service (MPS) enabled.

If MPS is not enabled, however, only the following APIs are supported:

- Point-to-point RMA
- `nvshmem_barrier_all()` host
- `nvshmemx_barrier_all_on_stream()`
- `nvshmem_sync_all()` host
- `nvshmemx_sync_all_on_stream()`

To enable complete NVSHMEM MPG support, the NVIDIA MPS server must be installed and be running on the system. To enable support for the complete API, the MPS server must also be configured to place a limit on the total GPU utilization of a maximum of 100%.

The NVSHMEM library will automatically detect when it runs on a system with more processes than GPUs and fan out the processes accordingly. It also automatically detects the presence of the MPS server daemon and GPU utilization configuration and enables the APIs accordingly. If an unsupported API is used in a limited MPG run, an error message will be printed, and the application will exit.

3.3.4. Using NVSHMEM With Your C or C++ Program

Procedure

1. Include `nvshmem.h` and `nvshmemx.h` from `include/`.
2. Point to the `include/` and `lib/` paths.
3. NVSHMEM users: If your C or C++ program only uses NVSHMEM, install Hydra Process Manager using the `install_hydra.sh` bash script under the `scripts/` directory.
   a). Provide the download and install location as arguments, for example:
   ```
   ./install_hydra.sh <download_path> <install_path>
   ```
   b). To run the NVSHMEM job, use `nvshmrun` launcher, which is located under `bin/` in the Hydra install path.

3.3.5. Using NVSHMEM With Your MPI or OpenSHMEM Program

Here is some information about how to use NVSHMEM with your MPI or OpenSHMEM program.

Note: The only currently tested MPI library is Open MPI, but any standard compliant MPI library should work.

To run a Hybrid MPI + NVSHMEM program, use the `mpirun` launcher in the MPI installation.

Similarly, NVSHMEM can be used from OpenSHMEM programs, and you must use the corresponding launcher for the OpenSHMEM library. The only currently tested OpenSHMEM version is OSHMEM in Open MPI. Other OpenSHMEM implementations, such as Sandia OpenSHMEM (SOS) should also work, but these implementations have not been tested. To run the hybrid OpenSHMEM/NVSHMEM job, use the `oshrun` launcher in the OpenMPI installation or follow the launcher specification of your OpenSHMEM library.

NVSHMEM relies on a plug-in system for bootstrapping. By default, an MPI bootstrap plug-in is built for NVSHMEM and is installed in `$(NVSHMEM_HOME)/lib`. If this directory is not in your dynamic linker search path, you might need to add it to `$LD_LIBRARY_PATH`. This MPI plug-in is selected automatically at runtime if the `nvshmemx_init_attr` initialization function is used to request the MPI bootstrap, or if `NVSHMEM_BOOTSTRAP="MPI"` is set.

The source code of the MPI bootstrap plug-in is installed in `$(NVSHMEM_HOME)/share/nvshmem/src/bootstrap-plugins` and can be built separately from the NVSHMEM library (for example, to support additional MPI libraries). Custom bootstrap plugins are also possible and should implement the interface that is defined in `$(NVSHMEM_HOME)/include/nvshmem_bootstrap.h`. Plug-ins must be built as relocatable shared objects.

After the external plug-in library is built, it can be specified to NVSHMEM at runtime by specifying `NVSHMEM_BOOTSTRAP="plugin"` and `NVSHMEM_BOOTSTRAP_PLUGIN="[name of plugin]"`. For example, `NVSHMEM_BOOTSTRAP="MPI"` is equal to `NVSHMEM_BOOTSTRAP="plugin"` and `NVSHMEM_BOOTSTRAP_PLUGIN="nvshmem_bootstrap_mpi.so"`.

3.4. Running Performance Tests

Before you can run performance tests, you first must build them.

Procedure

1. If the NVSHMEM library was built with `NVSHMEM_MPI_SUPPORT=1`, set the `CUDA_HOME`, `NVSHMEM_HOME` and `MPI_HOME` environment variables to build NVSHMEM performance tests:

   ```
   CUDA_HOME=<path to supported CUDA installation>
   NVSHMEM_HOME=<path to directory where NVSHMEM is installed>
   MPI_HOME=<path to MPI installation>
   ```

   If you built NVSHMEM with MPI and OpenSHMEM support (`NVSHMEM_MPI_SUPPORT=1` and `NVSHMEM_SHMEM_SUPPORT=1`) when you build `perftest/`, MPI and OpenSHMEM support must be enabled.

   Build without SHMEM interoperability: To build NVSHMEM performance tests without SHMEM interoperability, set the environment variable `NVSHMEM_SHMEM_SUPPORT` to `0`. By default, performance tests are installed under `perftest/perftest_install`. To install to a different path, set `NVSHMEM_PERFTEST_INSTALL` to point to the correct path.
2. Update `LD_LIBRARY_PATH` to point to `$CUDA_HOME/lib64`, `$MPI_HOME/lib`, and `$NVSHMEM_HOME/lib`.
3. Assuming Hydra is installed under `HYDRA_HOME`, run performance tests as NVSHMEM jobs, hybrid MPI+NVSHMEM jobs, or hybrid OpenSHMEM+NVSHMEM jobs with the following commands (using `perftest/device/pt-to-pt/put.cu` as an example):

   NVSHMEM job using Hydra (PMI-1):

   ```
   $HYDRA_HOME/bin/nvshmrun -n <up to number of P2P or InfiniBand
   NIC accessible GPUs>
   $NVSHMEM_PERFTEST_INSTALL/device/pt-to-pt/shmem_put_bw
   ```

   NVSHMEM job using Slurm:

   ```
   srun -n <up to number of P2P or InfiniBand NIC accessible GPUs>
   $NVSHMEM_PERFTEST_INSTALL/device/pt-to-pt/shmem_put_bw
   ```

   > Note: When Slurm was built with a PMI that does not match the default of NVSHMEM, for example, if Slurm was built with PMIx support and `NVSHMEM_DEFAULT_PMIX=1` was not set when building NVSHMEM, `NVSHMEM_BOOTSTRAP_PMI` can be used to override the default. Possible values are PMIX, PMI-2, and PMI. The Slurm `--mpi=` option to srun can be used to tell Slurm which PMI interface to use.

   Hybrid MPI/NVSHMEM job:

   ```
   $MPI_HOME/bin/mpirun -n <up to number of GPUs accessible by P2P
   or InfiniBand NIC> -x NVSHMEMTEST_USE_MPI_LAUNCHER=1
   $NVSHMEM_PERFTEST_INSTALL/device/pt-to-pt/shmem_put_bw
   ```

   Hybrid OpenSHMEM/NVSHMEM job:

   ```
   $MPI_HOME/bin/oshrun -n <up to number of GPUs accessible by P2P
   or InfiniBand NIC> -x USE_SHMEM_IN_TEST=1
   $NVSHMEM_PERFTEST_INSTALL/device/pt-to-pt/shmem_put_bw
   ```

3.5. "Hello World" Example

Procedure

1. Save the following code as `nvshmemHelloWorld.cu`:

```
#include <stdio.h>
#include <cuda.h>
#include <nvshmem.h>
#include <nvshmemx.h>

__global__ void simple_shift(int *destination) {
    int mype = nvshmem_my_pe();
    int npes = nvshmem_n_pes();
    int peer = (mype + 1) % npes;

    nvshmem_int_p(destination, mype, peer);
}

int main(void) {
    int mype_node, msg;
    cudaStream_t stream;

    nvshmem_init();
    mype_node = nvshmem_team_my_pe(NVSHMEMX_TEAM_NODE);
    cudaSetDevice(mype_node);
    cudaStreamCreate(&stream);

    int *destination = (int *) nvshmem_malloc(sizeof(int));

    simple_shift<<<1, 1, 0, stream>>>(destination);
    nvshmemx_barrier_all_on_stream(stream);
    cudaMemcpyAsync(&msg, destination, sizeof(int), cudaMemcpyDeviceToHost,
 stream);

    cudaStreamSynchronize(stream);
    printf("%d: received message %d\n", nvshmem_my_pe(), msg);

    nvshmem_free(destination);
    nvshmem_finalize();
    return 0;
}
```

2. Build `nvshmemHelloWorld.cu` with the following command:

   When using dynamic linking:

```
nvcc -rdc=true -ccbin g++ -gencode=$NVCC_GENCODE -I
$NVSHMEM_HOME/include nvshmemHelloWorld.cu -o
nvshmemHelloWorld.out -L $NVSHMEM_HOME/lib -lnvshmem_host -lnvshmem_device
```

   When using static linking:

```
nvcc -rdc=true -ccbin g++ -gencode=$NVCC_GENCODE -I
$NVSHMEM_HOME/include nvshmemHelloWorld.cu -o
nvshmemHelloWorld.out -L $NVSHMEM_HOME/lib -lnvshmem -lnvidia-ml -lcuda -lcudart
```

   Where `arch=compute_70,code=sm_70` is the value of `NVCC_GENCODE` for V100 GPUs.

3. Run the `nvshmemHelloWorld` sample with one of the following commands:

   - When running on one host with two GPUs (connected by PCI-E, NVLink or Infiniband):

```
$HYDRA_HOME/bin/nvshmrun -n 2 -ppn 2 ./nvshmemHelloWorld.out
```

- When running on two hosts with one GPU per host that is connected by InfiniBand:

```
$HYDRA_HOME/bin/nvshmrun -n 2 -ppn 1 --hosts hostname1,hostname2 ./
nvshmemHelloWorld.out
```

Chapter 4. Support

Report bugs and submit feature requests by using NVONLINE or by emailing nvshmem@nvidia.com.

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.

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.

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.

VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

OpenCL

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

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
www.nvidia.com

Trademarks

NVIDIA, the NVIDIA logo, and CUDA, CUDA Toolkit, GPU, Kepler, Mellanox, NVLink, NVSHMEM, and Tesla are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2019-2023 NVIDIA Corporation and affiliates. All rights reserved.

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
www.nvidia.com