CUDA on WSL User Guide

User Guide

DG-05603-001_v11.6 | March 2022

Table of Contents

Chapter 1. What is WSL?

WSL or Windows Subsystem for Linux is a Windows feature that enables users to run native Linux applications, containers and command-line tools directly on Windows 11 and later OS builds.

Why WSL?

Some applications are available for installation on both Linux and Windows. But this is not always the case when it comes to technologies that had previously dominated one ecosystem. For instance, the data center and cloud services market are predominantly Linux driven, whereas, consumer desktops and laptops and enterprise systems are typically Windows. But this differentiation from a user perspective continues to fade as technological uses and applicability converge. Users want to browse and check emails, play games while also developing their cloud-native applications. Enterprise wants to have the IT management capabilities offered in Windows alongside Linux deployment capabilities.

Typically developers have the following means to work across both Linux and Windows applications

- Use different systems for Linux and Windows, or
- Dual Boot i.e. Install Linux and Windows in separate partitions on the same or different hard disks on the system and boot to the OS of choice.

These are not always suitable options as it impedes workflow and is not seamless. One has to stop all the work and then switch the system or reboot. Also this does not solve the problem of integrated workflow where the developers want to leverage tools and software systems across two dominant ecosystems.

Traditional Virtual Machines vs. WSL 2

Whether to efficiently use hardware resources or to improve productivity, virtualization is a more widely used solution in both consumer and enterprise space. There are different types of virtualizations, and it is beyond the scope of this document to delve into the specifics. But traditional virtualization solutions require installation and setup of a virtualization management software to manage the guest virtual machines.

Although WSL 2 is itself a Virtual Machine, unlike traditional VMs it is easy to setup as it is provided by the host operating system provider and is quite lightweight. Applications running within WSL see less overhead compared to traditional VMs especially if they require access to the hardware or perform privileged operations compared to when run directly on the system. This is especially important for GPU accelerated workloads.

While VMs allow applications to be run unmodified, due to constraints from setup and performance overhead they not the best option in many situations.

Containers vs. WSL 2

While a VM provides a secure self-contained, execution environment with a complete user space for the application, containers enable application composability without the overhead of VMs. Containers compose all the dependencies of the applications such as libraries, files etc., to be bundled together for development and easy and predictable deployment. Containers run on the operating system that is installed on the system directly and therefore do not provide full isolation from other containers like a VM does, but keeps overhead negligible as a result.

To learn more about differences between VMs and containers, see https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/containers-vs-vm.

WSL 1 vs. WSL 2

WSL2 is the second generation of WSL that offers the following benefits:

- Linux applications can run as is in WSL 2. WSL 2 is characteristically a VM with a Linux WSL Kernel in it that provides full compatibility with mainstream Linux kernel allowing support for native Linux applications including popular Linux distros.
- Faster file system support and that's more performant.
- WSL 2 is tightly integrated with the Microsoft Windows operating system, which allows it to run Linux applications alongside and even interop with other Windows desktop and modern store apps.

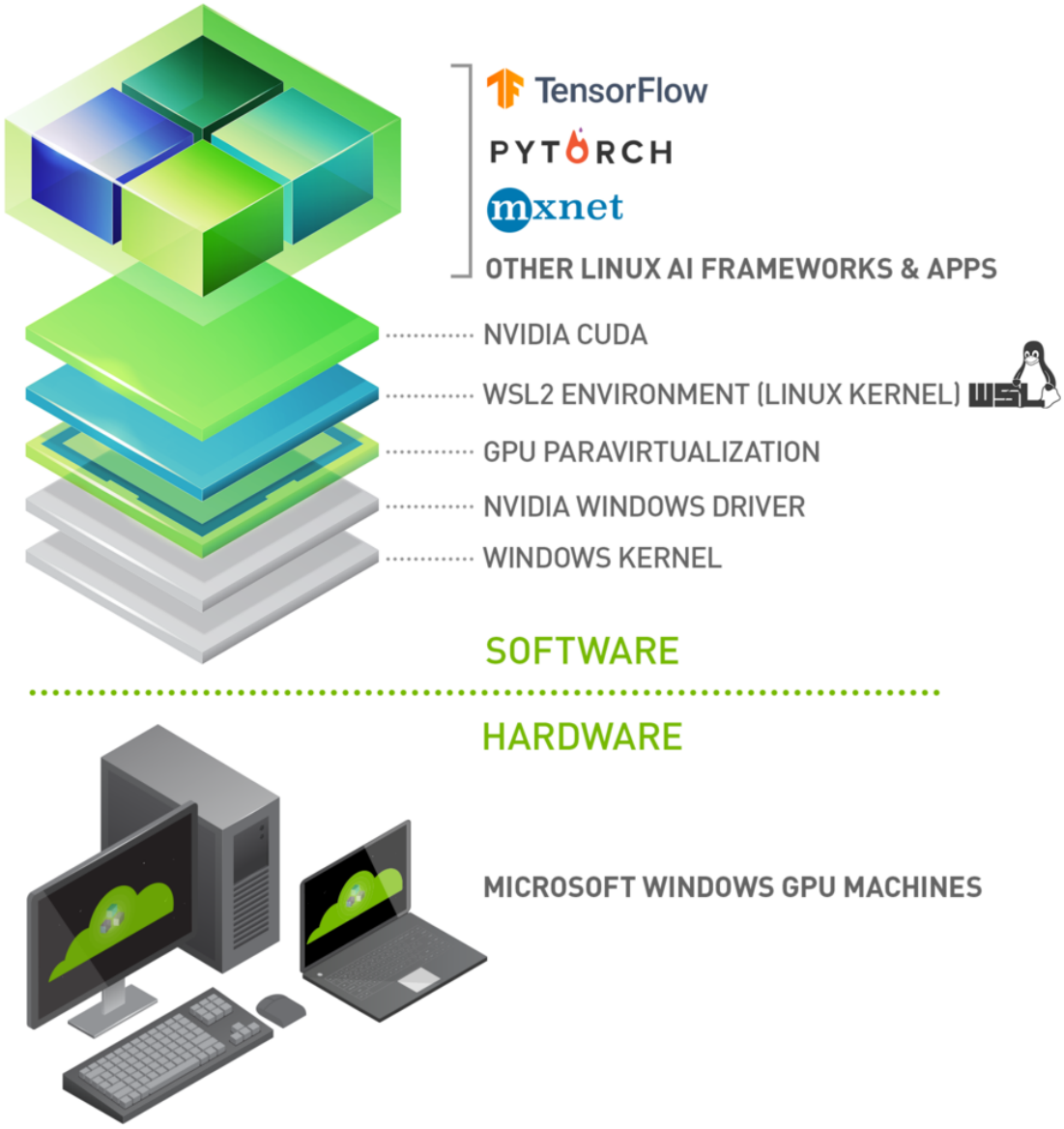
For the rest of this user guide WSL and WSL 2 may be used interchangeably.

Chapter 2. NVIDIA GPU Accelerated Computing on WSL 2

In WSL 2, Microsoft introduced GPU Paravirtualization Technology that, together with NVIDIA CUDA and other compute frameworks and technologies, makes GPU accelerated computing for data science, machine learning and inference solutions possible on WSL. GPU acceleration also serves to bring down the performance overhead of running an application inside a WSL-like environment close to near-native by being able to pipeline more parallel work on the GPU with less CPU intervention. NVIDIA driver support for WSL 2 does not stop with CUDA and associated compute software stack. There is DirectX support to enable graphics on WSL 2 by supporting DX12 APIs along with Direct ML support. For some helpful examples, see https://docs.microsoft.com/en-us/windows/win32/direct3d12/gpu-tensorflow-wsl.

WSL2 is a key enabler in making GPU acceleration to be seamlessly shared between Windows and Linux applications on the same system a reality. This offers flexibility and versatility while also serving to open up GPU accelerated computing by making it more accessible.

Figure 1. Illustration of the possibilities with NVIDIA CUDA software stack on WSL 2



This document describes a workflow for getting started with running CUDA applications or containers in a WSL 2 environment.

Chapter 3. Getting Started with WSL 2

To get started with running CUDA on WSL, complete these steps in order:

3.1. Step 1: Install NVIDIA Driver for GPU Support

- Download and install NVIDIA GeForce Game Ready or NVIDIA RTX Quadro Windows 11 display driver on your system with a compatible GeForce or NVIDIA RTX/Quadro card from

  https://developer.nvidia.com/cuda/wsl

Note: This is the only driver you need to install. Do not install any Linux display driver in WSL.

3.2. Step 2: Install WSL 2

1. Launch your preferred Windows Terminal / Command Prompt / Powershell and install WSL:

   ```
   wsl.exe --install
   ```

2. Ensure you have the latest WSL kernel:

   ```
   wsl.exe --update
   ```

3.3. Step 3: Set Up a Linux Development Environment

By default, WSL2 comes installed with Ubuntu. Other distros are available from the Microsoft Store.

From a Windows terminal, enter WSL:

```
wsl.exe
```

To update the distro to your favorite distro from the command line and to review other WSL commands, refer to the following resources:

- https://docs.microsoft.com/en-us/windows/wsl/install
- https://docs.microsoft.com/en-us/windows/wsl/basic-commands

Chapter 4. Getting Started with CUDA on WSL 2

CUDA support on WSL 2 allows you to run existing GPU accelerated Linux applications or containers such as RAPIDS or Deep Learning training or inference. If you are interested in building new CUDA applications, CUDA Toolkit must be installed in WSL.

4.1. Running Existing CUDA Applications

If your CUDA application binary is on Windows, the Windows `C:\` drive is already mounted to WSL 2 at `/mnt/c`.

Just copy the CUDA application over and run it:

```
cd /mnt/c/Users/<username>/Desktop

cp /mnt/c/Users/<username>/Desktop/CUDA-Samples/BlackScholes .

./BlackScholes
```

4.2. Running Existing GPU Accelerated Containers on WSL 2

This section describes the workflow for setting up the necessary software in WSL2 in preparation for running GPU accelerated containers.

4.2.1. Step 1: Install Docker

Use the Docker installation script to install standard Docker-CE for your choice of WSL 2 Linux distribution.

```
curl https://get.docker.com | sh
```

4.2.2. Step 2: Install NVIDIA Container Toolkit

Set up the `stable` repositories and the GPG key. The changes to the runtime to support WSL 2 are available in the `stable` repositories:

```
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)

curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -

curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list |
 sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```

Install the NVIDIA runtime packages (and their dependencies) after updating the package listing:

```
sudo apt-get update

sudo apt-get install -y nvidia-docker2
```

Open a separate WSL 2 window and start the Docker daemon again using the following commands to complete the installation:

```
sudo service docker stop

sudo service docker start
```

In the following section, we will walk through some examples of running containers in a WSL 2 environment.

4.2.3. Running Simple CUDA Containers

In this example, let's run an N-body simulation CUDA sample. This example has already been containerized and available from NGC.

```
docker run --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark
```

From the console, you should see output as shown below.

```
$ docker run --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -benchmark
Run "nbody -benchmark [-numbodies=<numBodies>]" to measure performance.
        -fullscreen       (run n-body simulation in fullscreen mode)
        -fp64             (use double precision floating point values for
 simulation)
        -hostmem          (stores simulation data in host memory)
        -benchmark        (run benchmark to measure performance)
        -numbodies=<N>    (number of bodies (>= 1) to run in simulation)
        -device=<d>       (where d=0,1,2.... for the CUDA device to use)
        -numdevices=<i>   (where i=(number of CUDA devices > 0) to use for
 simulation)
        -compare          (compares simulation results running once on the default
 GPU and once on the CPU)
        -cpu              (run n-body simulation on the CPU)
        -tipsy=<file.bin> (load a tipsy model file for simulation)

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary
 when GPU Boost is enabled.

> Windowed mode
```

```
> Simulation data stored in video memory
> Single precision floating point simulation
> 1 Devices used for simulation
GPU Device 0: "Turing" with compute capability 7.5

> Compute 7.5 CUDA device: [NVIDIA GeForce RTX 2080]
47104 bodies, total time for 10 iterations: 68.445 ms
= 324.169 billion interactions per second
= 6483.371 single-precision GFLOP/s at 20 flops per interaction
```

4.2.4. Deep Learning Framework Containers

Get started quickly with AI training using pre-trained models available from NVIDIA and the NGC catalog. Follow the instructions in this post for more details.

As an example, let's run a TensorFlow container to do a ResNet-50 training run using GPUs using the 20.03 container from NGC.

```
$ docker run --gpus all -it --shm-size=1g --ulimit memlock=-1 --ulimit
 stack=67108864 nvcr.io/nvidia/tensorflow:20.03-tf2-py3

================
== TensorFlow ==
================

NVIDIA Release 20.03-tf2 (build 11026100)
TensorFlow Version 2.1.0

Container image Copyright (c) 2019, NVIDIA CORPORATION.  All rights reserved.
Copyright 2017-2019 The TensorFlow Authors.  All rights reserved.

Various files include modifications (c) NVIDIA CORPORATION.  All rights reserved.
NVIDIA modifications are covered by the license terms that apply to the underlying
 project or file.

NOTE: MOFED driver for multi-node communication was not detected.
      Multi-node communication performance may be reduced.

root@c64bb1f70737:/workspace# cd nvidia-examples/
root@c64bb1f70737:/workspace/nvidia-examples# ls
big_lstm  build_imagenet_data  cnn  tensorrt
root@c64bb1f70737:/workspace/nvidia-examples# python cnn/resnet.py
...
WARNING:tensorflow:Expected a shuffled dataset but input dataset `x` is not
 shuffled. Please invoke `shuffle()` on input dataset.
2020-06-15 00:01:49.476393: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
2020-06-15 00:01:49.701149: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcudnn.so.7
global_step: 10 images_per_sec: 93.2
global_step: 20 images_per_sec: 276.8
global_step: 30 images_per_sec: 276.4
```

4.2.5. Jupyter Notebooks

In this example, let's run a Jupyter notebook.

```
docker run -it --gpus all -p 8888:8888 tensorflow/tensorflow:latest-gpu-py3-jupyter
```

After the container starts, you should see that everything is ready:

```
_________                                _________________
```

```
___     __/_____________________________________     ____/__  /________      __
__  /  _  _ \_  __ \_  ___/  __ \_  ___/_  /_   __  /_  __ \_ | /| / /
_  /   /  __/  / / /(__  )/ /_/ /  /   _  __/   _  / / /_/ /_ |/ |/ /
/_/    \___//_/ /_//____/ \____//_/    /_/      /_/  \____/____/|__/

WARNING: You are running this container as root, which can cause new files in
mounted volumes to be created as the root user on your host machine.

To avoid this, run the container by specifying your user's userid:

$ docker run -u $(id -u):$(id -g) args...

[I 04:00:11.167 NotebookApp] Writing notebook server cookie secret to /root/.local/
share/jupyter/runtime/notebook_cookie_secret
jupyter_http_over_ws extension initialized. Listening on /http_over_websocket
[I 04:00:11.447 NotebookApp] Serving notebooks from local directory: /tf
[I 04:00:11.447 NotebookApp] The Jupyter Notebook is running at:
[I 04:00:11.447 NotebookApp] http://72b6a6dfac02:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b
[I 04:00:11.447 NotebookApp]  or http://127.0.0.1:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b
[I 04:00:11.447 NotebookApp] Use Control-C to stop this server and shut down all
 kernels (twice to skip confirmation).
[C 04:00:11.451 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///root/.local/share/jupyter/runtime/nbserver-1-open.html
    Or copy and paste one of these URLs:
        http://72b6a6dfac02:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b
     or http://127.0.0.1:8888/?
token=6f8af846634535243512de1c0b5721e6350d7dbdbd5e4a1b
```

After the URL is available from the console output, input the URL into your browser to start developing with the Jupyter notebook. Ensure that you replace `127.0.0.1` with `localhost` in the URL when connecting to the Jupyter notebook from the browser. You should be able to view the TensorFlow tutorial in your browser. Choose any of the tutorials for this example.

Navigate to the **Cell** menu and select the **Run All** item, then check the log within the Jupyter notebook WSL 2 container to see the work accelerated by the GPU of your Windows PC.

```
...
[I 16:00:00.150 NotebookApp] 302 GET /?
token=1fc498fd08ea697cd1a01b5061bcc1b381254eeb4f768d5d (172.17.0.1) 0.51ms
[I 16:00:07.509 NotebookApp] Writing notebook-signing key to /root/.local/share/
jupyter/notebook_secret
[W 16:00:07.511 NotebookApp] Notebook tensorflow-tutorials/classification.ipynb is
 not trusted

[I 16:00:08.281 NotebookApp] Kernel started: 8fcadc98-b40e-41ee-8ce2-003afd444678
2021-07-07 16:00:14.343853: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libnvinfer.so.6
2021-07-07 16:00:14.357695: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libnvinfer_plugin.so.6
2021-07-07 16:00:25.573058: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcuda.so.1
2021-07-07 16:00:25.736913: E tensorflow/stream_executor/cuda/
cuda_gpu_executor.cc:967] could not open file to read NUMA node: /sys/bus/pci/
devices/0000:65:00.0/numa_node
Your kernel may have been built without NUMA support.

...
2021-07-07 16:00:26.149385: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcudart.so.10.1
```

```
2021-07-07 16:00:27.309024: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1096]
 Device interconnect StreamExecutor with strength 1 edge matrix:
2021-07-07 16:00:27.309079: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1102]
      0
2021-07-07 16:00:27.309101: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1115]
 0:   N
2021-07-07 16:00:27.310983: E tensorflow/stream_executor/cuda/
cuda_gpu_executor.cc:967] could not open file to read NUMA node: /sys/bus/pci/
devices/0000:65:00.0/numa_node
Your kernel may have been built without NUMA support.
2021-07-07 16:00:27.311180: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1324]
 Could not identify NUMA node of platform GPU id 0, defaulting to 0.  Your kernel
 may not have been built with NUMA support.
2021-07-07 16:00:27.311599: E tensorflow/stream_executor/cuda/
cuda_gpu_executor.cc:967] could not open file to read NUMA node: /sys/bus/pci/
devices/0000:65:00.0/numa_node
Your kernel may have been built without NUMA support.
2021-07-07 16:00:27.311963: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1241]
 Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 6706
 MB memory) -> physical GPU (device: 0, name: NVIDIA GeForce RTX 2080, pci bus id:
 0000:65:00.0, compute capability: 7.5)
2021-07-07 16:00:30.450162: W tensorflow/core/framework/cpu_allocator_impl.cc:81]
 Allocation of 376320000 exceeds 10% of system memory.
2021-07-07 16:00:31.962248: I tensorflow/stream_executor/platform/default/
dso_loader.cc:44] Successfully opened dynamic library libcublas.so.10
2021-07-07 16:01:15.050709: W tensorflow/core/framework/cpu_allocator_impl.cc:81]
 Allocation of 62720000 exceeds 10% of system memory.
2021-07-07 16:01:16.025014: W tensorflow/core/framework/cpu_allocator_impl.cc:81]
 Allocation of 62720000 exceeds 10% of system memory.
[I 16:02:09.048 NotebookApp] Saving file at /tensorflow-tutorials/
classification.ipynb
```

4.2.6. Building Your Own GPU-accelerated Application on WSL 2

In order to compile a CUDA application on WSL 2, you will have to install the CUDA toolkit for Linux from the WSL prompt.

Normally, CUDA toolkit for Linux will have the CUDA driver for NVIDIA GPU packaged with it. On WSL 2, the CUDA driver used is part of the Windows driver installed on the system and therefore care must be taken to not install this Linux driver as it will clobber your installation.

We recommend either of the following two options for installing the CUDA Toolkit.

Note: The software versions are examples only. In the command below, use the CUDA and driver versions that you want to install.

Option 1: Using the WSL-Ubuntu Package

Launch WSL 2:

```
C:\> wsl
```

Install CUDA:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/wsl-ubuntu/x86_64/
cuda-wsl-ubuntu.pin
sudo mv cuda-wsl-ubuntu.pin /etc/apt/preferences.d/cuda-repository-pin-600
wget https://developer.download.nvidia.com/compute/cuda/11.4.0/local_installers/
cuda-repo-wsl-ubuntu-11-4-local_11.4.0-1_amd64.deb
```

```
sudo dpkg -i cuda-repo-wsl-ubuntu-11-4-local_11.4.0-1_amd64.deb
sudo apt-key add /var/cuda-repo-wsl-ubuntu-11-4-local/7fa2af80.pub
sudo apt-get update
sudo apt-get -y install cuda
```

Option 2: Using the Meta Package

If you installed the toolkit using the WSL-Ubuntu package, please skip this section. Meta packages do not contain the driver, so by following the steps below, you will be able to get just the CUDA toolkit installed on WSL.

Launch WSL 2:

```
C:\> wsl
```

Install CUDA:

```
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin
sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
wget https://developer.download.nvidia.com/compute/cuda/11.4.0/local_installers/cuda-repo-ubuntu2004-11-4-local_11.4.0-470.42.01-1_amd64.deb
sudo dpkg -i cuda-repo-ubuntu2004-11-4-local_11.4.0-470.42.01-1_amd64.deb
sudo apt-key add /var/cuda-repo-ubuntu2004-11-4-local/7fa2af80.pub
sudo apt-get update
```

Do not choose the “cuda”, “cuda-11-0”, or “cuda-drivers” meta-packages under WSL 2 as these packages will result in an attempt to install the Linux NVIDIA driver under WSL 2.

```
apt-get install -y cuda-toolkit-11-4
```

You can also install other components of the toolkit by choosing the right meta-package.

4.2.7. Building CUDA Samples

Build the CUDA samples available from GitHub or the ones under `/usr/local/cuda/samples` from your installation of the CUDA Toolkit in the previous section. The BlackScholes application is located under `/usr/local/cuda/samples/4_Finance/BlackScholes`. Alternatively, as mentioned earlier you can transfer a binary built on Linux to WSL 2 and run directly.

```
C:\> wsl
To run a command as administrator (user “root”), use “sudo <command>”.
See “man sudo_root” for details.
$ cd /usr/local/cuda-11.4/samples/4_Finance/BlackScholes
$ make BlackScholes
$ ./BlackScholes

Initializing data...
...allocating CPU memory for options.
...allocating GPU memory for options.
...generating input data in CPU mem.
...copying input data to GPU mem.
Data init done.

Executing Black-Scholes GPU kernel (131072 iterations)...
Options count             : 8000000
BlackScholesGPU() time    : 0.207633 msec
Effective memory bandwidth: 385.295561 GB/s
Gigaoptions per second    : 38.529556
```

Chapter 5. WSL 2 System Requirements

- WSL 2 GPU acceleration will be available on Pascal and later GPU architecture on both GeForce and Quadro product SKUs in WDDM mode. It will not be available on Quadro GPUs in TCC mode or Tesla GPUs yet.
- Ensure you are on the latest WSL Kernel or at least 4.19.121+. Once again we recommend 5.10.16.3 or later for better performance and functional fixes.
- If you are on Windows 11, you no longer need to be on Windows Insider Program to use WSL. Refer to Windows11 system requirements.
- If you are continuing to use Windows 10, see Windows Insider Preview and Windows 10 Support.

Chapter 6. NVIDIA Compute Software Support Matrix for WSL 2

Package	Suggested Versions
NVIDIA Windows Driver x86 - CUDA, Video Nvidia-smi (Limited Feature Set)	R495 and later Windows production drivers will officially support WSL 2 on Pascal and later GPUs. For the latest features, use the WSL 2 driver published on CUDA Developer Zone.
NVIDIA Container Toolkit	Minimum versions - v2.6.0 with libnvidia-container - 1.5.1+
CUDA toolkit	Latest CUDA toolkit from 11.x releases can be used. Developer tools: Debuggers and Profilers are not supported yet.
CUDA Developer Tools	Compute Sanitizer - Pascal and later Nsight Systems CLI, and CUPTI (Trace) - Volta and later
RAPIDS	21.10 - Experimental Support for single GPU
NCCL	2.11.4+

Chapter 7. Known Limitations for Linux CUDA Applications

The following table lists the known limitations on WSL 2 that may affect CUDA applications that use some of these features that are fully supported on Linux.

Limitations	Impact
Maxwell GPU is not supported.	Maxwell GPUs are not officially supported in WSL 2, but it may still work. Pascal and later GPU is recommended.
Unified Memory - Full Managed Memory Support is not available on Windows native and therefore WSL 2 will not support it for the foreseeable future.	UVM full features will not be available and therefore applications relying on UVM full features may not work. If your application is using Managed Memory, your application could see reduced performance and high system memory usage. Concurrent CPU/GPU access is not supported. CUDA queries will say whether it is supported or not and applications are expected to check this.
Pinned system memory (example: System memory that an application makes resident for GPU accesses) availability for applications is limited.	For example, some deep learning training workloads, depending on the framework, model and dataset size used, can exceed this limit and may not work.
Root user on bare metal (not containers) will not find nvidia-smi at the expected location.	Use `/usr/lib/wsl/lib/nvidia-smi` or manually add `/usr/lib/wsl/lib/` to the PATH).
With the NVIDIA Container Toolkit for Docker 19.03, only `--gpus all` is supported.	On multi-GPU systems it is not possible to filter for specific GPU devices by using specific index numbers to enumerate GPUs.

Chapter 8. Features Not Yet Supported

The following table lists the set of features that are currently not supported.

Limitations	Impact
Legacy CUDA IPC APIs are not yet supported. Newer style cumemmap IPC with fd is supported.	Typically multi-process / multi-gpu applications will be impacted.
CUDA Developer tools - Debuggers and Profilers	Developers who require debugging support are encouraged to find alternatives in the meanwhile.
NVML (nvidia-smi) does not support all the queries yet.	GPU utilization, active compute process are some queries that are not yet supported. Modifiable state features (ECC, Compute mode, Persistence mode) will not be supported.
OpenGL-CUDA Interop is not yet supported.	Applications relying on OpenGL will not work.

Chapter 9. Windows Insider Preview and Windows 10 Support

- If you are on Windows 11 please skip this section. Windows 11 is generally available to the public and therefore does not require special registration. All the instructions at the beginning of this user guide were mainly focused toward Windows 11 users.
- If you are looking to use WSL 2 on Windows 10 or to be on the bleeding edge of WSL 2 development, you may want to register for the Windows Insider Program and choose the appropriate flighting channel (previously fast rings) and get the latest build for your needs.
- Learn more on Releasing Windows 10 Build 19043.1263 (21H1) to Release Preview Channel.
- You can check your build version number by running `winver` via the Run command.

Chapter 10. Troubleshooting

10.1. Container Runtime Initialization Errors

In some cases, when running a Docker container, you may encounter `nvidia-container-cli : initialization error`:

```
$ sudo docker run --gpus all nvcr.io/nvidia/k8s/cuda-sample:nbody nbody -gpu -
benchmark
docker: Error response from daemon: OCI runtime create failed:
 container_linux.go:349: starting container process caused "process_linux.go:449:
 container init caused \"process_linux.go:432: running prestart hook 0 caused
 \\\"error running hook: exit status 1, stdout: , stderr: nvidia-container-cli:
 initialization error: driver error: failed to process request\\\\n\\\"\"": unknown.
ERRO[0000] error waiting for container: context canceled
```

This usually indicates that the right Windows OS build or Microsoft Windows Insider Preview Builds (Windows 10 only), WSL 2, NVIDIA drivers and NVIDIA Container Toolkit may not be installed correctly. Review the known issues and changelog sections to ensure the right versions of the driver and container toolkit are installed.

Ensure you have followed through the steps listed under Setup under Running CUDA containers; especially ensure that the docker daemon is still running.

```
sudo service docker stop
sudo service docker start
```

Or start the daemon directly and see if that resolves the issue:

```
sudo dockerd
```

If you are still running into this issue, use the `dxdiag` tools from the Run dialog and provide the diagnostic logs to NVIDIA by posting in the Developer Forums or by filing a report.

You can also use the CUDA on WSL 2 Developer Forums to get in touch with NVIDIA product and engineering teams for help.

10.2. Checking WSL Kernel Version

1. Ensure you have the latest kernel by running the following command in PowerShell:

```
$ wsl cat /proc/version

Linux version 5.10.16.3-microsoft-standard-WSL2
(x86_64-msft-linux-gcc (GCC) 9.3.0, GNU ld (GNU Binutils) 2.34.0.20200220) #1 SMP
 Fri Apr 2 22:23:49 UTC 2021
```

2. If you don't have the latest WSL kernel, you will see the following blocking warning upon trying to launch a Linux distribution within the WSL 2 container:

Chapter 11. Release Notes

11.1. 510.06

11.1.1. Changelog

- 8/31/2021:
  - NVIDIA Driver for Windows 10 and Windows 11
  - WIP Build: 22000.160 on devchannel, 19044.1200 on release preview channel
  - WSL Linux Kernel 5.10.43

11.1.2. New Features

The following new features are included in this release:

- CUDA on WSL2 support for Windows 10 21H2 that is available on the release preview channel. https://blogs.windows.com/windowsexperience/2021/07/15/introducing-the-next-feature-update-to-windows-10-21h2/
- Added the following support with containers:
  - NVML/nvidia-smi
  - Windows 10 21H2 in release preview channel

    Users are recommended to install `nvidia-docker2` and its dependencies by enabling the `experimental` branch of the repos. After installation, verify that the `libnvidia-container` package is at least 1.5.0-rc.2 either using `apt list | grep -in libnvidia-container` or `nvidia-container-cli --version`

    See https://docs.nvidia.com/cuda/wsl-user-guide/index.html#ch05-running-containers
  - Users can also now query the GPU architecture in plain text using "`nvidia-smi -q`"

```
==============NVSMI LOG==============
Driver Version                            : 4XXX
CUDA Version                              : 11.X
```

```
Attached GPUs                             : 1
GPU 00000000:XX:00.0
Product Name                          : NVIDIA RTX XXXXX
Product Brand                         : NVIDIA RTX
Product Architecture                  : Ampere

<...>
```

11.1.3. Resolved Issues

With the above latest nvidia-docker2 from the experimental branch, the following issue is resolved:

- When running the NGC Deep Learning (DL) Framework GPU containers in WSL 2, you will no longer encounter the below message:

```
The NVIDIA Driver was not detected.  GPU functionality will not be
    available.
```

11.1.4. Known Limitations

The following features are not supported in this release:

1. Note that NVIDIA Container Toolkit has not yet been validated with Docker Desktop WSL 2 backend. Use Docker-CE for Linux instead inside your WSL 2 Linux distribution.
2. CUDA debugging or profiling tools are not supported in WSL 2. This capability will be added in a future release.
3. `cumemmap` IPC with fd is now supported. Other Legacy IPC APIs are not yet supported.
4. Unified Memory is limited to the same feature set as on native Windows systems.
5. With the NVIDIA Container Toolkit for Docker 19.03, only `--gpus all` is supported. This means that on multi-GPU systems it is not possible to filter for specific GPU devices by using specific index numbers to enumerate GPUs.

11.1.5. Known Issues

The following are known issues in this release:

- "`/usr/lib/wsl/lib`" is not included in the WSL PATH environment for OS Win10 21H2 (19044.1200).

  To use nvidia-smi on Win 10 21H1(19044.1200), manually add "`/usr/lib/wsl/lib`" to the PATH or invoke the nvidia-smi tool within WSL specifying the full path as `/usr/lib/wsl/lib/nvidia-smi`.

11.2. 471.21

Changelog

- 7/14/2021:

- NVIDIA Driver for Windows 10 and later: 471.21
- Windows 11 officially supported
- WIP build: 22000, WSL Linux Kernel 5.10.43

New Features

The following new features are included in this release:

- None.

Known Limitations

The following features are not supported in this release:

- Note that NVIDIA Container Toolkit has not yet been validated with Docker Desktop WSL 2 backend. Use Docker-CE for Linux instead inside your WSL 2 Linux distribution.
- CUDA debugging or profiling tools are not supported in WSL 2. This capability will be added in a future release.
- `cumemmap` IPC with fd is now supported. Other Legacy IPC APIs are not yet supported.
- Unified Memory is limited to the same feature set as on native Windows systems.
- With the NVIDIA Container Toolkit for Docker 19.03, only `--gpus all` is supported. This means that on multi-GPU systems it is not possible to filter for specific GPU devices by using specific index numbers to enumerate GPUs.
- When running the NGC Deep Learning (DL) Framework GPU containers in WSL 2, you may encounter a message:

```
The NVIDIA Driver was not detected.  GPU functionality will not be
 available.
```

  Note that this message is an incorrect warning for WSL 2 and will be fixed in future releases of the DL Framework containers to correctly detect the NVIDIA GPUs. The DL Framework containers will still continue to be accelerated using CUDA on WSL 2.

Known Issues

The following are known issues in this release:

- In-game Vsync fails to cap the frames to the refresh rate of the display only when Gsync is enabled. This will impact DX in-game vsync scenarios on native Windows with Gsync.

11.3. 470.76

- 6/3/2021:
  - nvidia-smi is enabled with limited feature support

- Fixes in the NVML libraries: Symbols not exported correctly in the NVML Lib preventing the library to be loaded in some cases
- Some performance optimization in the CUDA driver related to memory allocation
- Bug fixes in the CUDA driver for WSL
- NVIDIA Driver for Windows 10: 470.76
- WIP build: 21390, WSL Linux Kernel: 5.10.16

11.4. 470.14

- 3/23/2021: Resolved issues with nvidia-smi crashing on some systems.
  - NVIDIA Driver for Windows 10: 470.14
  - WIP build: 21332, WSL Linux kernel 5.4.91
- 3/9/2021
  - WIP OS 21313+ and Linux kernel 5.4.91
  - Do not use WIP OS 21327

11.5. 465.42

- 1/28/2021: CUDA Toolkit 11.2 support, nvidia-smi, NVML support and critical performance improvements.

  The following software versions are supported with this preview release for WSL 2:
  - NVIDIA Driver for Windows 10: 465.42
  - Recommended WIP build: 21292

11.6. 465.21

- 12/16/2020: Support for 3060Ti. Fix for installation problems observed in notebooks

  The following software versions are supported with this preview release for WSL 2:
  - NVIDIA Driver for Windows 10: 465.21

11.7. 465.12

- 11/16/2020: The following software versions are supported with this preview release for WSL 2:
  - NVIDIA Driver for Windows 10: 465.12

11.8. 460.20

- 9/23/2020: The following software versions are supported with this preview release for WSL 2:
  - NVIDIA Driver for Windows 10: 460.20

11.9. 460.15

- 9/2/2020:

  The following software versions are supported with this preview release for WSL 2:
  - NVIDIA Driver for Windows 10: 460.15

11.10. 455.41

- 6/19/2020: Updated driver release to address cache coherency issues on some CPU systems, including AMD Ryzen.

  The following software versions are supported with this preview release for WSL 2:
  - NVIDIA Driver for Windows 10: 455.41

11.11. 455.38

- 6/17/2020: Initial Version.

  The following software versions are supported with this preview release for WSL 2:
  - NVIDIA Driver for Windows 10: 455.38
  - NVIDIA Container Toolkit: nvidia-docker2 (2.3) and libnvidia-container (>= 1.2.0-rc.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.

OpenCL

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

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

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