OpenLB technical report: Installing CUDA for OpenLB

Maximilian Schecher

March 2023

Installing CUDA for Nvidia GPUs

The following is a quick guide on how to install the CUDA functionality for Nvidia graphics cards on both Windows or Linux. The first two sections describe how to install CUDA on Windows via WSL or Linux, respectively. The third section discusses how to set up OpenMPI, a CUDA-aware MPI implementation, and the fourth and final section explains how to configure OpenLB to make use of the installed functionalities.

CUDA on Windows with WSL

The preferred approach for OpenLB on Windows is to use the Windows Subsystem for Linux (WSL). The following was written with the assumption that OpenLB has been successfully set up on WSL with Ubuntu.

The following specifications are needed to get CUDA running via WSL:

- Windows 10 version 21H2 or higher
- CUDA compatible Nvidia graphics card
- WSL 2 with a `glibc-based` distribution (e.g. Ubuntu)

To find out which Windows version exactly you're using, open up the `run` dialog box in Windows and type in the command `winver`, which will display a pop-up window similar to the one below:

Figure 1: Pop-up window displaying the exact version and build of Windows.

In order to find out what graphics card you have and whether it is compatible with CUDA, open the up the Windows `run` dialog and type in the command `dxdiag`, which will open the `DirectX Diagnostic Tool`. Under the tab `Render`, it will display the information regarding your graphics card. In the example picture of the `DirectX Diagnostic Tool` below, the graphics card is a `NVIDIA GeForce GTX 1650`. Nvidia provides the information on which graphics card is compatible with CUDA on their website (https://developer.nvidia.com/cuda-gpus).

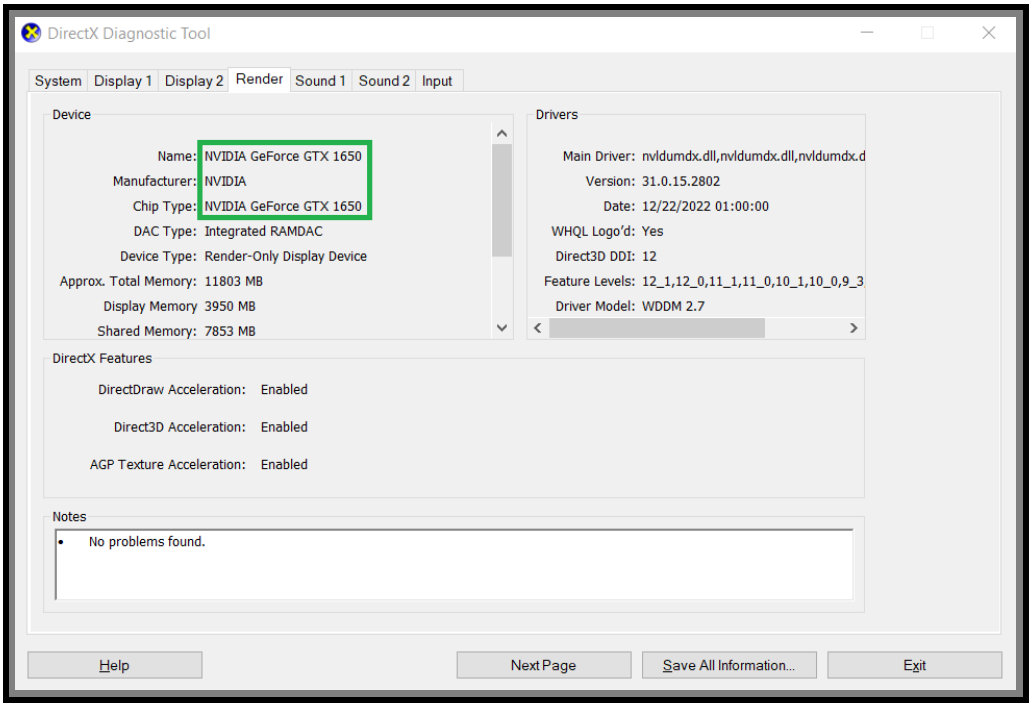


Figure 2: The `Render` tab of the `DirectX Diagnostic Tool`.

CUDA is only supported on version 2 of the Windows Subsystem for Linux (WSL). To confirm which version of WSL is installed, open the Windows PowerShell with administrator rights and type in the command

```
wsl --list --verbose
```

This will display which Linux distribution and which version of WSL is currently installed. The output should look similar to the following:

```
PS C:\Windows> wsl --list --verbose
  NAME      STATE           VERSION
* Ubuntu    Stopped         1
```

Listing 1: WSL table of installed distributions

In this example the distribution that is installed is `Ubuntu` and the WSL version is 1. Upgrading to the necessary version 2 can be done by typing

```
wsl --set-version Ubuntu 2
```

into the `PowerShell` terminal. Note that when using a different distribution for WSL, the command has to be adjusted accordingly.

An error might occur claiming that a certain hard-link target does not exist. This means that there is software installed on WSL that collides with the update. The error message will provide the path of the non-existing hard-link, which will be a hint onto which package causes this error. In the WSL terminal, the command

```
sudo apt list --installed
```

will give an overview over all the installed packages. The conflicting package can then be removed with

```
sudo apt-get remove [PACKAGE-NAME]
```

Once the package has been removed, WSL can be upgraded. On a succussful upgrade, we should receive a message that the conversion is complete and we can verify the version with the

```
wsl --list --verbose
```

command. The conflicting package can then be reinstalled.

In order for WSL to have access to the GPU hardware, virtual GPU needs to be enabled on Windows. This can be done by installing an appropriate driver on Windows. It should not be necessary to install any device drivers on WSL itself. It is even highly suggested not to do so, since any installation of a driver on WSL itself might override the functionality provided by the driver that is installed onto Windows. As of the writing of this guide, the most recent Nvidia drivers automatically support virtual GPU for WSL. The newest driver can be directly downloaded from the Nvidia website (https://www.nvidia.com/download/index.aspx). The website offers dropdown lists to specify what product type, device, operating system, etc. the driver is needed for. Once the most recent driver is installed, we can install the CUDA toolkit on WSL.

The following commands typed into the WSL terminal will install the Nvidia CUDA toolkit on WSL (Ubuntu):

```
sudo apt-key del 7fa2af80

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

sudo apt-key adv --fetch-keys https://developer.download.nvidia.com/
                               compute/cuda/repos/wsl-ubuntu/
                               x86_64/3bf863cc.pub

sudo add-apt-repository 'deb https://developer.download.nvidia.com/
                            compute/cuda/repos/wsl-ubuntu/x86_64/ /'

sudo apt-get update

sudo apt-get -y install cuda
```

Listing 2: Commands to install CUDA on WSL

If the Nvidia CUDA compiler is correctly installed, the command

```
nvcc --version
```

will reply with a message similar to the following:

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Mon_Oct_24_19:12:58_PDT_2022
Cuda compilation tools, release 12.0, V12.0.76
Build cuda_12.0.r12.0/compiler.31968024_0
```

Listing 3: Version details of an installed Cuda compiler

To check the versions of CUDA and the driver, the command

```
nvidia-smi
```

will respond with the Nvidia System Management Interface, displaying various information about the installed GPUs (see Figure 3). The CUDA toolkit should now be properly installed and working.

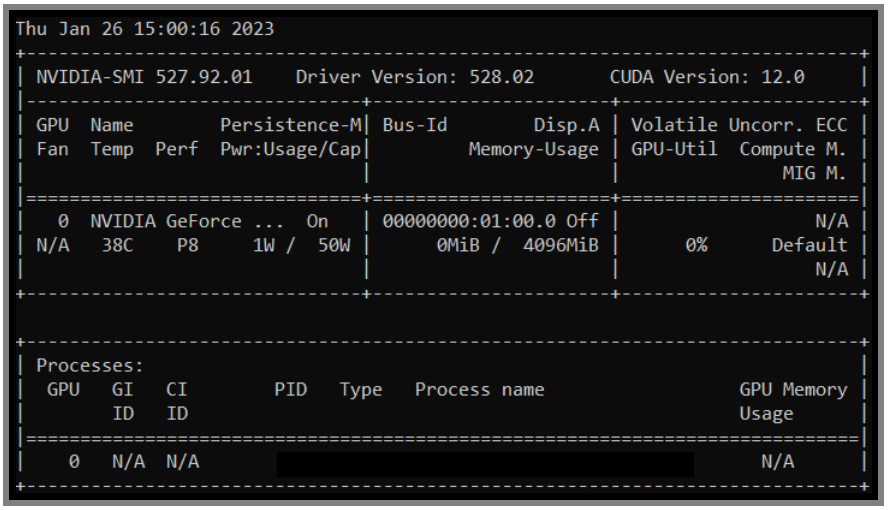


Figure 3: The NVIDIA System Management Interface

CUDA on Linux

Before installing the CUDA toolkit on Linux, typing the command

```
lspci | grep -i nvidia
```

can confirm that the GPU is CUDA-capable.

To install the CUDA toolkit on Linux, visit the the Nvidia website and choose the fitting operating system, architecture, distribution, as well as the preferred installation type for your system (https://developer.nvidia.com/cuda-toolkit). The website will then provide you with the correct commands with which you can install the CUDA toolkit on your Linux system.

After the installation of the toolkit, the environment variables need to be set:

```
export PATH=/usr/local/cuda-12.0/bin${PATH:+:${PATH}}
```

If the installation was done with a runfile, the `LD_LIBRARY_PATH` variable has to be set, as well. The following command sets this variable on a 64-bit system. The command for 32-bit systems is almost identical: `lib64` has to be exchanged for `lib`:

```
export LD_LIBRARY_PATH=/usr/local/cuda-12.0/lib64\
                       ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

If a different install path or version of the CUDA toolkit has been chosen during the installation process, both commands above have to be altered accordingly. To confirm that the installation has been successful, use the commands

```
nvcc --version
```

and

```
nvidia-smi
```

If the CUDA toolkit has been installed correctly, an output similar to those shown in Listing 3 and Figure 3 respectively.

OpenMPI

To have the functionality of MPI in combination with CUDA, there are several CUDA-aware MPI implementations available. This section will describe the installation of the open-source implementation OpenMPI in four steps:

1. Download the desired OpenMPI version from the website (https://www.open-mpi.org/software/). As of the writing of this guide, the most current version was `openmpi-4.1.5.tar.bz2`

2. In your Linux (or WSL for Windows) terminal, move to the folder where the file was saved to and extract the downloaded package via the command

```
tar -jxf openmpi-4.1.5.tar.bz2
```

3. Change into this new directory to configure, compile and install OpenMPI with the following three commands:

```
./configure --prefix=$HOME/opt/openmpi
            --with-cuda=/usr/local/cuda-12.0/include
make all
make install
```

Note that the path following `-prefix=` is the path we wish to install openmpi in and the path following `-with-cuda=` is the location of the `include` folder of your CUDA installation. These paths might be different depending on the users choices.

4. Change the environment variables with the following two commands in the Linux or WSL terminal:

```
echo "export PATH=\$PATH:\$HOME/opt/openmpi/bin" >> $HOME/.bashrc
echo "export LD_LIBRARY_PATH=\$LD_LIBRARY_PATH:
            \$HOME/opt/openmpi/lib" \ >> $HOME/.bashrc
```

Once again the path for openmpi, might be different, depending on where the software was installed.

To see whether the installation of OpenMPI was successful, we can enter the command

```
ompi_info --parsable -l 9
          --all | grep mpi_built_with_cuda_support:value
```

If the installation was done successfully, the terminal should respond with the output `true`.

Utilizing CUDA in OpenLB

The root directory contains a folder named `config`, in which several build config examples can be found. The `config.mk` makefile of the root directory can be replaced with the makefile that suits the current needs (e.g. using only the GPU, using the GPU with MPI, using CPU with MPI, etc.). Each example makefile also includes instructions.

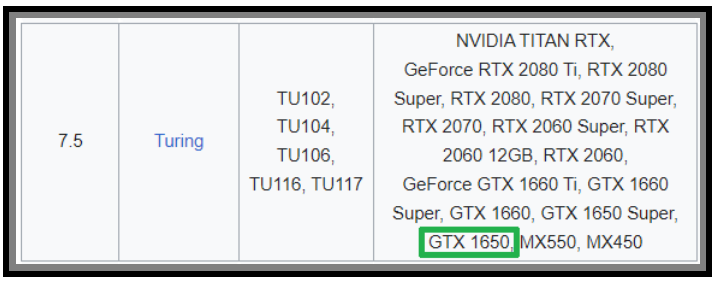
Make a backup of the current `config.mk` in the root directory and replace it with a copy of the makefile `gpu_only` found in the `config` folder. After renaming `gpu_only` to `config.mk`, we open the file and check the value of `CUDA_ARCH`: This value might have to be changed, depending on your graphics card and its architecture. The file `rules.mk` in the root directory contains a table that shows which architecture goes with which value:

```
## | CUDA Architecture | Version     |
## |-------------------+-------------|
## | Fermi             | 20          |
## | Kepler            | 30, 35, 37  |
## | Maxwell           | 50, 52, 53  |
## | Pascal            | 60, 61, 62  |
## | Volta             | 70, 72      |
## | Turing            | 75          |
## | Ampere            | 80, 86, 87  |
```

Listing 4: CUDA architectures and their corresponding version numbers

Another table on the internet (https://en.wikipedia.org/wiki/CUDA) shows which graphics card corresponds to which architecture. This guide used the

`GTX 1650` as an example for the graphics card. The following picture shows that the `GTX 1650` corresponds to the `Turing` architecture, so the value of `CUDA_ARCH` has to be set to 75 in both `config.mk` and `rules.mk` files. Af-



7.5	Turing	TU102, TU104, TU106, TU116, TU117	NVIDIA TITAN RTX, GeForce RTX 2080 Ti, RTX 2080 Super, RTX 2080, RTX 2070 Super, RTX 2070, RTX 2060 Super, RTX 2060 12GB, RTX 2060, GeForce GTX 1660 Ti, GTX 1660 Super, GTX 1660, GTX 1650 Super, GTX 1650, MX550, MX450

Figure 4: Table containing Nvidia GPUs with the `Turing` Microarchitecture.

ter saving the changes of `CUDA_ARCH` in both `config.mk` and `rules.mk`, the `config.mk` can be compiled via the command `make clean; make` in your WSL (or Linux) terminal.

It is now possible to compile and execute one of the GPU-enabled OpenLB examples with CUDA support.