UNIVERSITÄT TÜBINGEN
Prof. Dr.-Ing. Hendrik P.A. Lensch
Lehrstuhl für Computergrafik
Lukas Ruppert (lukas.ruppert@uni-tuebingen.de)
Faezeh Zakeri (faezeh-sadat.zakeri@uni-tuebingen.de)

February 20, 2024

Massively Parallel Computing
Assignment 0

Since we're running the exercises online again this year, there are a few setup steps you need to go through first before starting with the actual exercises. Please let us know if you have any questions or suggestions.

In principle, you can also run the exercises on your own machine if you have a CUDA-capable graphics card (i.e., one from NVIDIA). However, we can *not* support your individual environments on your machines.

0.1 Basic Setup

Please follow the instructions in the Computer Graphics Getting Started Guide to setup a SSH and VNC connection to our machines.

0.2 CUDA Programming Environment

In order to compile and run CUDA applications, you will need to install the CUDA Toolkit.

To test your CUDA environment, try running the NVIDIA CUDA Compiler `nvcc`:

```
$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Sep_21_10:33:58_PDT_2022
Cuda compilation tools, release 11.8, V11.8.89
Build cuda_11.8.r11.8/compiler.31833905_0
```

Also check your NVIDIA driver and CUDA Toolkit versions using `nvidia-smi`:

```
$ nvidia-smi
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.147.05   Driver Version: 525.147.05   CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
...
```

0.2.1 Choosing an IDE

For developing CUDA applications, you can choose between a few IDEs:

- Eclipse with the Nsight plugins.

  Eclipse is not widely used for C++ development (anymore), but used to be the basis of NVIDIA's official Nsight IDE (now deprecated). Thus, it has the best integrated debugger for CUDA which provides a nice overview of all running CUDA threads and allows to easily switch between them.

  CMake support is bad (system includes are not being found and, thus, code indexing fails). Instead, it depends on its own build system or manually written `Makefiles`.

Make sure to *exactly* follow the steps shown below on how to open/create CUDA projects in Eclipse.

When installing Eclipse version 2022-12 or later, make sure to use the plugins from CUDA Toolkit version 12.3 or later. Otherwise, you will not be able to run the debugger.

- Visual Studio Code with the Nsight Visual Studio Code Edition extension in addition to the usual C/C++ setup (C/C++, clangd, CMake extensions).

  Debug configurations need to be created manually as `launch.json` files. Make sure to set `"cwd"` to the absolute path of the project, `"program"` to the absolute path of the compiled program, and `"args"` to the program arguments. You can start with a copy of this example:

  ```
  {
      "version": "0.2.0",
      "configurations": [
          {
              "name": "Example Debug Configuration",
              "type": "cuda-gdb",
              "request": "launch",
              "cwd": "${workspaceFolder}",
              "program": "${workspaceFolder}/build/testGamma",
              "args": "vase.ppm 2.2 vase_gamma.ppm"
          }
      ]
  }
  ```

  (See the Variable Reference)

  While it is possible to debug CUDA threads, switching between them requires entering the thread and block ids manually and there is no overview of running threads or blocks like in Eclipse.

- QtCreator

  Enable the experimental clangd mode for the C++ code model. Debugging with `cuda-gdb` is possible, but the IDE does not know about GPU threads. (You can probably enter the necessary `cuda-gdb` commands manually in the debugging console if you really want to do that...)

- Kate

  Sometimes, a simple text editor is all you need. It has support for `clangd` (you might have to enable the LSP plugin in the settings). So, if you're looking for something small, this might be the right tool for you.

- CLion

  Supports CUDA. Only free to use with a student's license – you're on your own.

When using VS Code, QtCreator, or Kate with `clangd`, consider creating a file `~/.config/clangd/config.yaml` with the following content

```
If:
    PathMatch: [.*\.cu, .*\.cuh]
CompileFlags:
    Add: [ -xcuda ]
    Remove: [ --generate-code=*, -forward-unknown-to-host-compiler ]
```

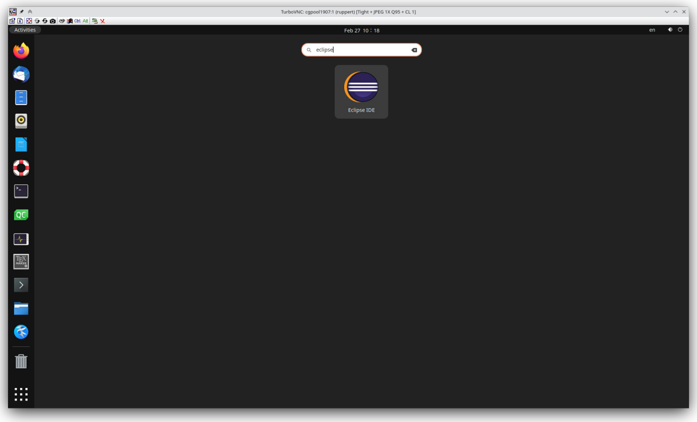
This forces `clangd` to interpret all `*.cu` and `*.cuh` files as CUDA source files and tells it to ignore some compiler flags specific to `nvcc`.

If you are unsure about what to choose, try using Eclipse as shown below.

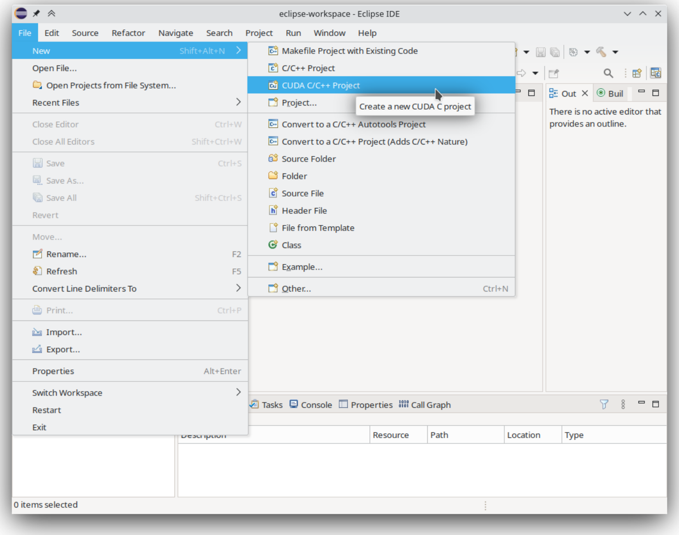
0.3 Import, Compile and Run a Sample CUDA Program

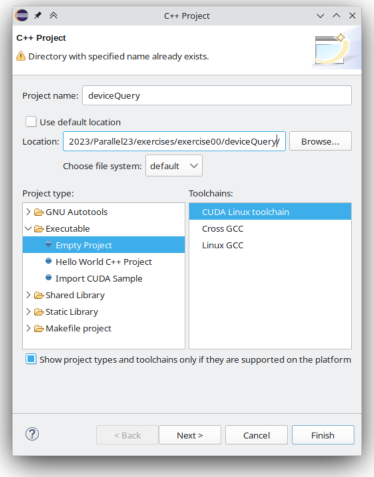
Download `exercise00.zip` containing the `deviceQuery` sample application from ILIAS and open it in Eclipse. See the following figures for details on how to import the project, compile, and run it.

First, launch Eclipse from the desktop's application launcher, or open a terminal and run `eclipse`.



Then, create an new *CUDA C/C++ Project*. (If it does not show up in the menu, choose Other... and select it from the C/C++ folder.) Set the path to the folder containing the sample project and type in a project name. Press "Finish".




C++ Project

Directory with specified name already exists.

Project name: deviceQuery

Use default location: ☐

Location: 2023/Parallel23/exercises/exercise00/deviceQuery/

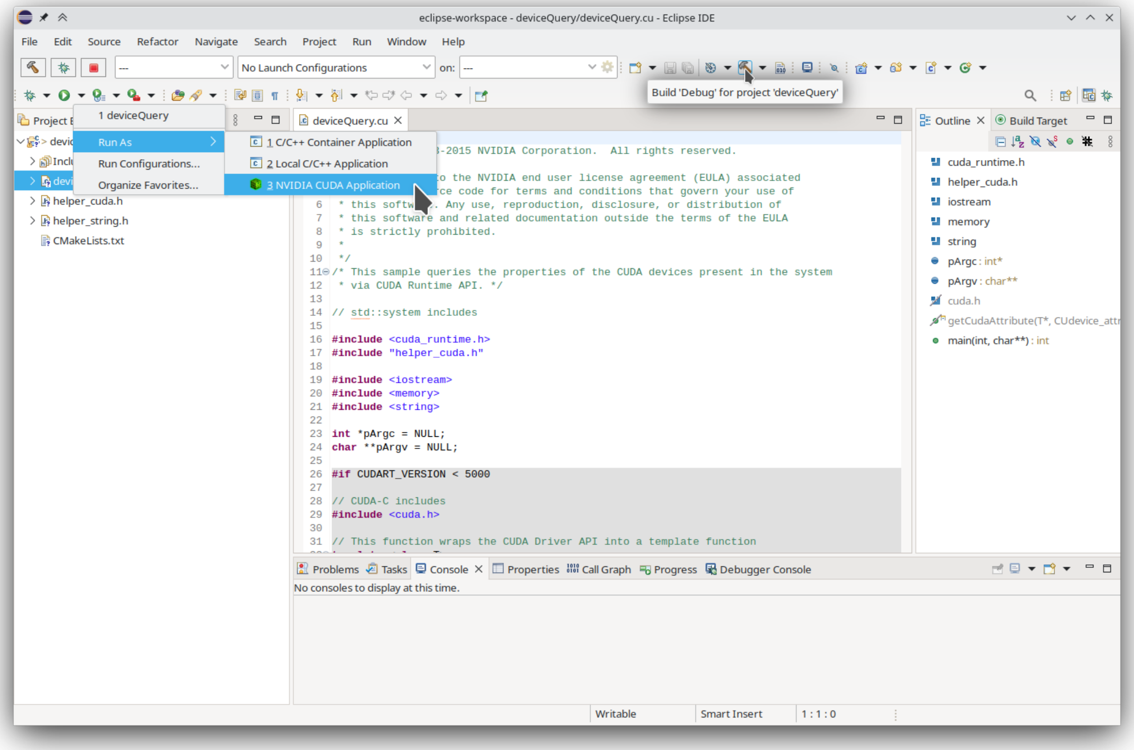
Choose file system: default

Project type: Executable › Empty Project

Toolchains: CUDA Linux toolchain

Show project types and toolchains only if they are supported on the platform: ☑

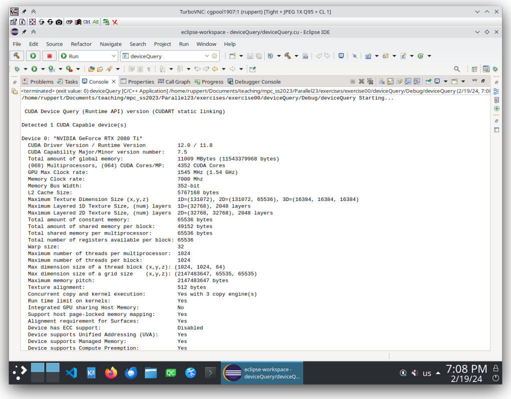
Build the project using the hammer icon. Then, run it as an *NVIDIA CUDA Application*. For running it again, you can just press the play button rather than using the drop-down menu.
(Avoid the icons with a border around them on the top left. They ignore the current file and are related to the drop-down menus next to them. The icons without a border around them will always build and run the project containing the current file.)



```
-2015 NVIDIA Corporation.  All rights reserved.

o the NVIDIA end user license agreement (EULA) associated
ce code for terms and conditions that govern your use of
 * this softw   . Any use, reproduction, disclosure, or distribution of
 * this software and related documentation outside the terms of the EULA
 * is strictly prohibited.
 *
 */
/* This sample queries the properties of the CUDA devices present in the system
 * via CUDA Runtime API. */

// std::system includes

#include <cuda_runtime.h>
#include "helper_cuda.h"

#include <iostream>
#include <memory>
#include <string>

int *pArgc = NULL;
char **pArgv = NULL;

#if CUDART_VERSION < 5000

// CUDA-C includes
#include <cuda.h>

// This function wraps the CUDA Driver API into a template function
```

```
No consoles to display at this time.
```

You should get the following output:


TurboVNC: cgpool1907:1 (ruppert) [Tight + JPEG 1X Q95 + CL 1]
eclipse-workspace - deviceQuery/deviceQuery.cu - Eclipse IDE
Run
deviceQuery
Problems
Tasks
Console
Properties
Call Graph
Progress
Debugger Console
eclipse-workspace - deviceQuery/deviceQ...
7:08 PM
2/19/24

```
<terminated> (exit value: 0) deviceQuery [C/C++ Application] /home/ruppert/Documents/teaching/mpc_ss2023/Parallel23/exercises/exercise00/deviceQuery/Debug/deviceQuery (2/19/24, 7:0
/home/ruppert/Documents/teaching/mpc_ss2023/Parallel23/exercises/exercise00/deviceQuery/Debug/deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce RTX 2080 Ti"
  CUDA Driver Version / Runtime Version          12.0 / 11.8
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 11009 MBytes (11543379968 bytes)
  (068) Multiprocessors, (064) CUDA Cores/MP:    4352 CUDA Cores
  GPU Max Clock rate:                            1545 MHz (1.54 GHz)
  Memory Clock rate:                             7000 Mhz
  Memory Bus Width:                              352-bit
  L2 Cache Size:                                 5767168 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 3 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            Yes
```

0.3.1 Using only SSH.

If you prefer not using VNC and just want to use SSH instead, you can also compile and run the deviceQuery application by running the following commands in the project folder:

```
# this should work for all CMake-based projects
mkdir build
cd build
cmake ..
make
```

Then, run the compiled program using ./deviceQuery.