ADASS GPU Tutorial 2024

Keith Shortridge, K&V, 1 October 2024

Introduction

This tutorial is aimed at ADASS attendees who may have sat through numerous talks about how GPUs make everything faster, wondered about making use of their GPUs to speed up compute tasks, and then somehow never found the time to try it. The aim is to give people who have no experience with GPU programming a kick-start towards trying it for themselves on their own laptops. The tutorial will be based around a small set of example C++ command line programs that perform calculations on 2D data, all of which run on MacOS and/or Linux. Attendees will be able to build, run, modify, and experiment with these programs, seeing how the GPU performance compares with the CPU. These instructions describe how to download and build these example programs.

You need to do the following:

- Download the source code for the programs.
- If you don't have them on your machine already, install the libraries they use. This should be straightforward.
- Build the programs – this should amount to no more than typing 'make' in a terminal window.
- Come to the tutorial and bring your laptop.

(Note: The tutorial is designed for MacOS and Linux. Interested Windows users should read the section "For Windows Users".)

Downloading the source code.

The source code is available on GitHub at:

https://github.com/KnaveAndVarlet/ADASS2024_GPU

Look at the right-hand column under 'Releases' and click on the latest release. Download the .zip or .tar.gz file and extract the contents.

This gives you a directory called ADASS2024_GPU-<version>. In it there are two sub-directories, Metal and Vulkan, containing the source files for the Metal and Vulkan version of

the example programs. Metal is Apple's GPU programming API, and Vulkan is a cross-platform GPU API from the Khronos group. Metal only runs on Apple devices, while Vulkan should be supported on almost any modern GPU.

Metal and Vulkan have three sub-directories, Adder, Median, and Mandel, containing the three example programs. Each of these individual program sub-directories is self-contained., with its own copies of any common source files.

Installing the necessary libraries and tools.

Detailed instructions for those that need them are in the system-dependent appendices that follow, but this section provides an overview. You don't need much.

Because both Metal and Vulkan versions of the programs are available, you need to decide which you want to work with.

In most cases, this decision is already made for you. If you are using a Mac, Metal is the obvious choice (although you can use Vukan, or can even try both). If you are using Linux, you need to use Vulkan.

Programming tools

You will need:

- A C++17 compliant C++ compiler and the associated run-time libraries, and 'make'.
- A source code editor that you're comfortable with. You'll be looking at code with a lot of comments, so one with syntax highlighting will help.

If you don't have the programming tools installed, Linux users will find details at the start of the appendices on installing Vulkan on Ubuntu or Fedora. Mac users should check the appendix "Installing programming tools on MacOS".

GPU Libraries

Metal users will need the Metal libraries and files, but these should be already installed on a Mac. You should have all you need.

Vulkan users (usually under Linux), need Vulkan itself, together with the GLFW window library and the GLM linear algebra library. You will also need a compiler for the glslc shader language. See the appendices "Installing Vulkan etc on Ubuntu" and "Installing Vulkan etc on Fedora". Mac users who want to use Vulkan instead of/as well as Metal should see the appendix "Installing Vulkan etc on MacOS".

Cfitsio and FITS files

One of the examples, Median, works with FITS files, and you will need the cfitsio library. This is a standard building block for astronomical software, and you probably already have it. If

not, see the appendix “Installing Cfitsio”. It will help if you have a FITS image viewer such as SAOImageDS9 available, just to be able to look at the images produced by the program. There is one small test FITS image included, but you may like to have some data of your own to try. Any FITS file whose main image is a reasonably-sized 2D image should be suitable.

Installation checklist.

You should have:

- The source code for the example programs.
- A C++ compiler good for C++17, that will run as c++ from the command line. (For Mac users working with metal this needs to be Mac’s standard clang C++ compiler.)
- A version of ‘make’ that runs from the command line.
- The cfitsio library and associated include files.
- Linux users (and Mac users working with Vulkan) need the Vulkan, GLFW and GLM libraries and associated include files, and a glslc compiler.
- Ideally, a FITS viewer such as SAOImageDS9.

Building the example programs.

Once you’ve downloaded the source code, and have the necessary libraries installed, building the example programs should be simple. All you need to do is go into Vulkan or Metal directory and type ‘make’. So, starting from the top-level ADASS2024_GPU directory:

```
cd Vulkan
```

Or:

```
cd Metal
```

then

```
make
```

The resulting executable programs are called Adder, Median, and Mandel, and are in the sub-directories of the same name. They need to be run from those sub-directories. You can see if they run – Adder and Median should just run and output something to the effect that everything ran OK, while Mandel should open a graphics window and draw a fairly well-known shape in it. If that doesn’t happen, let me know (again, e-mail to keith@knaveandvarlet.com.au). You can play with them more if you like, but that might spoil any surprise the tutorial has in store.

Possible problems

If you have any problems building the programs, some of the libraries or their associated include files may be in a non-standard location and the compiler or linker isn't finding them. You may find you have to modify the Makefile(s) to allow for this. All the Makefiles start by setting values for INCLUDES and LIBRARIES, and you may need to add directories to these. Do feel free to e-mail keith@knaveandvarlet.com.au preferably with as many details (error messages copied from the terminal, for example) as possible.

If any of the programs build but don't run, they may not be able to find all the shared libraries. On Linux, the command "`ldd <program name>`" will show which shareable libraries the program is using, and if it can find them or not. The MacOS equivalent is "`otool -L <program name>`". You many find you need to add a non-standard directory to the LD_LIBRARY_PATH or DYLD_LIBRARY_PATH environment variables. Again, feel free to e-mail for help.

On some systems, a Vulkan program will run but will produce a warning of the form "Warning: terminator_CreateInstance: Failed to CreateInstance in ICD 0. Skipping ICD." When it starts up, Vulkan looks in /usr/share/vulkan/icd.d/ for a set of .json files listing the available implementations. In this case, it's found it can't use one of these. You can force it to use a specific file by setting VK_ICD_FILENAMES appropriately, eg:

```
export VK_ICD_FILENAMES=/usr/share/vulkan/icd.d/radeon_icd.x86_64.json
```

which forces use of the specific ICD for an AMD Radeon GPU.

It's also always worth checking that your Linux graphics drivers are up to date.

Help, support and feedback

Any problems, e-mail keith@knaveandvarlet.com.au

If you do hit a problem and fix it yourself, please let me know the details. Someone else may hit the same problem and this will help me to help them. And please feel free to tell me if you got things to run without problems; it's always nice to know!

There may well be minor corrections and changes to the example programs between now and the actual tutorial, in which case new releases should appear on GitHub.

For Windows Users

The example programs to be used in the tutorial have been developed and tested using MacOS and Linux. Vulkan programs run perfectly well under Windows – a test version of one of the example programs has been built experimentally as a native Windows program using Virtual Studio – but the build process is rather different, and the examples as released have

some ancillary code that is not Windows-compatible. However, it is possible to build the Vulkan versions of the programs under Windows using WSL (Windows Subsystem for Linux), which runs Ubuntu in a virtual machine under Windows.

If you have WSL and Ubuntu installed on your Windows laptop, which is the default setup produced by the `wsl --install` command described in https://learn.microsoft.com/en-us/windows/wsl/install, the various programming tools and libraries can be installed exactly as described for Ubuntu. (You may find it easiest to download the tar.gz files used for the glslc compiler and the code for the example programs using Windows and to copy the .tar.gz files over to the Ubuntu disk using File Explorer.) The example programs should build and run as described in the rest of this document.

The problem with this is that WSL does not provide access to the actual GPU hardware. The programs run, but they use Mesa's Lavapipe to emulate the GPU in software. It's a very impressive emulation, perfectly usable, and would allow you to follow along with the tutorial, but you won't see the speed increase that the actual GPU would provide.

(You will also get the "Warning: terminator_CreateInstance: Failed to CreateInstance in ICD 0. Skipping ICD." warning described above under Possible Problems. In this case, setting:

```
export VK_ICD_FILENAMES=/usr/share/vulkan/icd.d/lvp_icd.x86_64.json
```

will get rid of the warning.)

Appendix: Installing Vulkan etc on Ubuntu

These instructions are a simplified version of more detailed instructions from the excellent vulkan-tutorial site. They should work on Ubuntu with the apt package manager. A version for Fedora and dnf can be found in the following appendix. For more details:

https://vulkan-tutorial.com/Development_environment

o Make sure things are up to date:

```
sudo apt update
```

o Install the standard C++ programming tools, if you need them:

```
sudo apt install build-essential
```

o Install Vulkan

```
sudo apt install vulkan-tools
```

(at this point, it might be as well to see if Vulkan is supported on your machine - try the 'vulkaninfo' command - you should see a lot of detail you don't care about, but it should work - and then try the 'vkcube' command to see if you see a spinning cube).

o Install Vulkan development tools:

```
sudo apt install libvulkan-dev
sudo apt install vulkan-validationlayers-dev
```

o Install GLFW and GLM:

```
sudo apt install libglfw3-dev
sudo apt install libglm-dev
```

o Install Google's glslc shader compiler. Go to:

```
https://github.com/google/shaderc/blob/main/downloads.md
```

and download the gcc binary version for Linux. (Or the clang version, if that's your preferred C++ compiler.) Uncompress it and copy over the compiler. Assuming your downloads go into ~/Downloads:

```
cd ~/Downloads
tar -xvf install.tgz
sudo cp install/bin/glslc /usr/local/bin
```

Appendix: Installing Vulkan etc on Fedora

These instructions are a simplified version of more detailed instructions from the excellent vulkan-tutorial site. They should work on Fedora with the dnf package manager. For more details:

https://vulkan-tutorial.com/Development_environment

o Make sure things are up to date:

```
sudo dnf check-update
sudo dnf upgrade
```

o Install the standard C++ programming tools, if you need them:

```
sudo dnf install gcc-c++
sudo dnf install make
```

o Install Vulkan

```
sudo dnf install vulkan-tools
```

(at this point, it might be as well to see if Vulkan is supported on your machine - try the 'vulkaninfo' command - you should see a lot of detail you don't care about, but it should work - and then try the 'vkcube' command to see if you see a spinning cube).

o Install Vulkan development tools:

```
sudo dnf install vulkan-loader-devel
sudo dnf install mesa-vulkan-devel
sudo dnf install vulkan-validation-layers-devel
```

o Install GLFW and GLM:

```
sudo dnf install glfw-devel
sudo dnf install glm-devel
```

o Install Google's glslc shader compiler.

```
sudo dnf install glslc
```

(Or you can download it directly from the Google web pages as described in the Ubuntu appendix.)

Appendix: Installing Cfitsio

Cfitsio is a standard library for astronomical software, and you probably already have it on your laptop. If not, the traditional way to install it on a UNIX-like system (including MacOS) is to download the source and build it using a 'configure', 'make' sequence.

Cfitsio is available through the usual package managers. On MacOS you can use homebrew (`brew install cfitsio`). Ubuntu has the package 'libcfitsio-dev' (`sudo apt install libcfitsio-dev`), and Fedora has 'cfitsio-devel' (`sudo dnf install libcfitsio-dev`)

(Alternatively, the cfitsio source can be downloaded from NASA's HEASARC page:

https://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html

which has a link in the first section called 'cfitsio_latest.tar.gz' which will download the latest version, which can be installed using the usual `./configure, make, sudo make install` sequence. However, cfitsio also uses the curl library for network file access, and the zlib library for handling compressed files. If you don't have these on your machine, you can disable the use of curl by adding `–disable-curl` to the 'configure' line, but Zlib will have to be downloaded and installed. At the end of the Zlib home page https://zlib.net there is a permalink for the most recent Zlib version. Download this, uncompress it, and build it again with `./configure, make, sudo make install.` You may want to control the location these are installed using, for example, `–prefix=/usr/local` in the configure line, and you may need to make sure the libraries are picked up by adding, for example, `/usr/local` to `LD_LIBRARY_PATH` or `DYLD_LIBRARY_PATH`.)

Appendix: Installing programming tools on MacOS

If you're a programmer using a Mac, you probably already have Apple's Xcode development environment installed. The example programs for this tutorial are set up as traditional command line programs built using Makefiles, not as Xcode projects, so you don't need Xcode as such, but it's the most straightforward way to get the necessary development tools, especially as the Makefiles assume you have the clang compiler, which is the default C++ compiler used by Apple.

If you don't have Xcode, open the App Store application (in the Applications folder – it's also a standard Dock item), search for Xcode and install it. The command line versions of the development tools are not installed by default, but if you don't have them and you give a command like 'clang' or 'make' from the terminal, you should get a prompt asking if you want to install them. Alternatively, you can install them directly from the terminal using the command 'xcode-select --install'.

Appendix: Installing Vulkan etc on MacOS

These instructions are a simplified version of more detailed instructions from the excellent vulkan-tutorial site. For more details:

https://vulkan-tutorial.com/Development_environment

o Install Vulkan

Download the latest version of the SDK installer for MacOS from the LunarG web page at:

https://vulkan.lunarg.com/sdk/home

and run it. When asked for options, select the System global installation option – you don't need any of the others. This will perform a full installation of Vulkan and the MoltenVk layer needed to run it on Macs and will put the necessary files in /usr/local – that's what the global installation option does. (MoltenVk is a layer that implements Vulkan on top of Metal.)

At this point, it might be as well to see if you can get Vulkan to display something – you should have a new VulkanSDK directory, with a subdirectory for the version of Vulkan installed, eg 1.3.290, which contains various sub-directories including macOS and Applications. Applications should contain vkcube.app; run that using '`open vkcube.app`' and you should see a spinning cube.

o Install GLFW and GLM:

The easiest way to install these is using Homebrew, in which case you need:

```
brew install glfw
```

```
brew install glm
```

(Depending on your setup, the Makefiles may or may not find the include files and libraries installed by Homebrew. I have had to add /opt/homebrew/include to CPATH and /opt/homebrew/lib to LIBRARY_PATH in my .zshrc file on some machines, and after upgrading to Sonoma I had to add:

```
DYLD_LIBRARY_PATH+=/usr/local/lib
export DYLD_LIBRARY_PATH
```

as well.)

o Install Google's glslc shader compiler. Go to:

https://github.com/google/shaderc/blob/main/downloads.md

and download the clang binary version for MacOS. Uncompress it and copy over the compiler as follows. Assuming your downloads go into ~/Downloads:

```
cd ~/Downloads
tar -xvf install.tgz
sudo cp install/bin/glslc /usr/local/bin
```