

How to set up OpenCL in Linux

You:

- Want to install OpenCL to compute on CPUs and GPUs

This guide:

- shows you how to easily set up a functional OpenCL environment on just about any Linux box

Contents

1. How to set up OpenCL in Linux
 1. Overview
 2. Installation
 1. Installing the AMD ICD loader and CPU ICD (from the "APP SDK")
 2. Installing the AMD ICD loader and CPU ICD (from the driver package-- probably unsupported)
 3. Using the AMD ICD with GPUs
 1. Running OpenCL code remotely on AMD GPUs (via SSH) without sudo
 1. Multiple GPUs
 4. Installing the Intel CPU ICD
 5. Installing the Nvidia ICD
 6. Debian
3. Tips and Tricks
 1. Per-user ICD registry?
 2. Downloading the OpenCL headers
4. Related Resources
5. Testing
 1. 1. Manual Method

Overview

To use OpenCL, you need three things:

libOpenCL.so -- the "ICD loader"

This library dispatches your OpenCL calls to OpenCL implementations. It is what you link into your OpenCL-using program (or into PyOpenCL, as the case

may be). Individual implementations are called ICDs ("Installable Client Drivers").

/etc/OpenCL/vendors/*.icd -- the "ICD registry"

These files tell the ICD loader what OpenCL implementations (ICDs) are installed on your machine. There's one for each ICD. Each file is a one-line text file containing the name of the dynamic library (aka shared object, aka ".so" file) containing the implementation. The single line may either be the full absolute path or just the file name, in which case the dynamic linker must be able to find that file--perhaps with the help of setting the `LD_LIBRARY_PATH` environment variable. The names of the `.icd` files themselves are arbitrary, but they must have a file extension of `.icd`.

One or more OpenCL implementations -- the "ICDs"


These are dynamic libraries pointed to by the `.icd` files, plus perhaps a few supporting files. They may reside anywhere on your file system. You can (and likely want to) have more than one installed.

Each implementation typically ships all of these components. They will tend to step on each other's feet if you're not careful. Avoiding such accidents is what this guide is about.


Installation

Installing the AMD ICD loader and CPU ICD (from the "APP SDK")

The AMD ICD works on both AMD and Intel CPUs. And is recommended even for Intel CPUs not providing SSE3/SEE4 support which is required by more recent versions of the Intel OpenCL SDK.

Update: as of July 2014, the AMD Accelerated Parallel Processing ("APP") package can be found  here. After agreeing to the license and downloading the appropriate TGZ file, untar it and run

```
sudo ./AMD-APP-SDK-v3.0.130.136-GA-linux64.sh
```


Reportedly, the instructions that used to be here no longer work. Greg von Winckel has written a  script that can be used instead to do the same thing.

If you want to build code using OpenCL, you probably also want the OpenCL headers.


See #headers.

Installing the AMD ICD loader and CPU ICD (from the driver package--probably unsupported)

The AMD ICD works on both AMD and Intel CPUs.

1. Download a driver from  this page.
2. Unzip the .zip file you got from the page.
3. Run

```
bash ./amd-driver-installer-catalyst-*.run --extract .
```

 Don't leave out the lone "." at the end.

4. For 64-bit:

```
TGT_DIR="/opt/amd-ocl-icd-VERSION.MINOR/lib"
mkdir -p "$TGT_DIR"
cp ./arch/x86_64/usr/lib64/* "$TGT_DIR"
echo "$TGT_DIR/libamdocl64.so" > /etc/OpenCL/vendors/amd.icd
```

For 32-bit:

```
TGT_DIR="/opt/amd-ocl-icd-VERSION.MINOR/lib"
mkdir -p "$TGT_DIR"
cp ./arch/x86/usr/lib/* "$TGT_DIR"
echo "$TGT_DIR/libamdocl32.so" > /etc/OpenCL/vendors/amd.icd
```

You probably want to change `VERSION.MINOR` to the version you actually downloaded. The `cp` might complain about leaving out 'fglrx', which is a directory--that's fine.

The `$TGT_DIR` will then contain the AMD ICD and the AMD ICD loader. If you want to build code using OpenCL, you probably also want the OpenCL headers. See #headers.

Using the AMD ICD with GPUs

Note that the above procedure *only* works for the CPU ICD. To use an AMD GPU, use the regular driver installation procedure of your Linux distribution, and do not follow the instructions above. The GPU driver will automatically install and set up everything

above.

Running OpenCL code remotely on AMD GPUs (via SSH) without sudo

To use an AMD GPU from OpenCL, the X server has to be running, and the OpenCL code has to have access to it. If you're running your code from within the X session, everything should just work. If you want to run code remotely, the following might help:

1. Create an **amdgpu** group, with all the users that need to run code remotely.
2. Create **/etc/enable-amd-compute** with the following contents:

```
#!/bin/bash
XAUTHPORT=$(echo $DISPLAY | cut -d'.' -f1 | cut -d':' -f2)
xauth extract /tmp/x11-auth-file ":$XAUTHPORT"
chmod 660 /tmp/x11-auth-file
chgrp amdgpu /tmp/x11-auth-file

cat > /tmp/enable-amd-compute <<EOF
export COMPUTE=$DISPLAY
unset DISPLAY
export XAUTHORITY=/tmp/x11-auth-file
EOF
```


3. Call **/etc/enable-amd-compute** from your X11 display manager init script. Just add

```
/etc/enable-amd-compute
```

to the end of **/etc/gdm3/Init/Default** for gdm3, or as


```
display-setup-script = /etc/enable-amd-compute
```

in the per-seat configuration for lightdm.

 Everyone in the **amdgpu** group now has full access to the X11 display. If someone else is doing sensitive work in the X session, remote users can spy on them.

Before running GPU computations remotely, you then have to run

```
source /tmp/enable-amd-compute
```

 For lightdm users, sometimes, just changing below variable(s) in **/etc/lightdm/lightdm.conf** is enough to use AMD ICD remotely.

```
xserver-allow-tcp=true  
autologin-user=USERNAME #optional
```

Multiple GPUs



To use multiple GPUs via OpenCL, use

```
aticonfig --initial --adapter=all
```

to create the Xorg configuration file.

Installing the Intel CPU ICD

 Intel offers CL downloads in multiple spots:

-  OpenCL driver download
-  Intel® SDK for OpenCL™ Applications

Of those, you want the "driver download" page.

1. There, grab the "OpenCL™ Runtime 15.1 for Intel® Core™ and Intel® Xeon® Processors for Red Hat* and SLES* Linux* OS (64-bit)" (does not just work on Red Hat/SLES)
2. Unpack: (your version number may vary)

```
tar xvf ~/Downloads  
/intel_sdk_for_ocl_applications_2013_xe_runtime_3.0.67279_x64.t  
gz
```

3. If your system is based on **rpm** (i.e. SuSE, Redhat, Fedora, CentOS, Mageia...), *and* if you want to install both Intel's ICD loader and ICD, simply type

```
cd intel_sdk_for*  
./install-cpu.sh
```

and you're done.

4. For other systems (Ubuntu, Debian, ...) and to install only the ICD, do the following:


```
TGT_DIR="/opt/intel-opencl-icd-VERSION.MINOR/lib"  
mkdir -p "$TGT_DIR"  
rpm2cpio intel_sdk_for_ocl_applications_*/opencl-*intel-  
cpu-*.x86_64.rpm | cpio -idmv  
cp ./opt/intel/opencl-*/lib64/* "$TGT_DIR"
```

```
echo "$TGT_DIR/libintelocl.so" > /etc/OpenCL/vendors/intel.icd
```

You probably want to change `VERSION.MINOR` to the version you actually downloaded. You also need an ICD loader. See the AMD instructions for that.


Installing the Nvidia ICD

Whatever procedure your distribution uses for installing the Nvidia GPU driver should automatically also install the OpenCL ICD. It might also overwrite your system-wide ICD loader, if you have one.

Note that you need the permission bits on `/dev/nvidia*` set to allow access if you want remote (SSH) GPU access. Since many Linux distributions play idiotic games with those permission bits as well as the ownership of the device, the most robust solution I've found is to use  POSIX ACLs:

```
setfacl -m g:gpu:rw /dev/nvidia*
```

Debian

 Debian packages OpenCL very well and makes installing this much easier.

Packages of ICD *loaders*: (you just need one of these)

- `amd-libopencl` (recommended)
- `nvidia-libopencl`
- `ocl-icd-libopencl` (the reference implementation)

Packages of ICDs:

- `amd-opencl-icd`
- `nvidia-opencl-icd`
- `beignet`

Package for headers:

- `opencl-headers`

I would recommend using the latest versions (perhaps from the 'experimental' repository). Those are usually the least buggy.

In particular, Debian makes it straightforward to use both AMD and Nvidia GPUs in the same computer. Just let the AMD driver drive the display, for Nvidia only install

the ICD.

The Intel CPU ICD is not packaged. Install it manually as above.

beignet is a project initiated by Intel to provide OpenCL support for their Ivy Bridge GPUs.

Tips and Tricks

Per-user ICD registry?

It's sometimes annoying that `/etc/OpenCL/vendors` is system-wide, i.e. individual users cannot install their own OpenCL ICDs. Here's a workaround.

1. Extract the AMD ICD (and ICD loader) as above, somewhere in your home directory.
2. Link against that `libOpenCL.so` when building your code.
3. Set `LD_LIBRARY_PATH` so that this `libOpenCL.so` gets picked up.
4. Make a new directory with a few `.icd` files as described above.
5. Set the environment variable `OPENCL_VENDOR_PATH` to that path.

Downloading the OpenCL headers

If you would also like the OpenCL headers, do the following:

```
TGT_DIR=/opt/ocl-headers/include/CL
mkdir -p $TGT_DIR && cd $TGT_DIR
wget https://raw.githubusercontent.com/KhronosGroup/OpenCL-Headers/master/ocl22/CL/{ocl,cl_platform,cl,cl_ext,cl_gl,cl_gl_ext}.h
```

Related Resources

-  Michael Hruby's guide to OpenCL on Ubuntu

Testing

```
apt-get install clinfo
```

```
clinfo
```


```

Number of platforms                                1
  Platform Name                                    NVIDIA CUDA
  Platform Vendor                                  NVIDIA Corporation
  Platform Version                                OpenCL 1.2 CUDA
8.0.0
  Platform Profile                                FULL_PROFILE
  Platform Extensions
cl_khr_global_int32_base_atomics
cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics
cl_khr_local_int32_extended_atomics cl_khr_fp64
cl_khr_byte_addressable_store cl_khr_icd cl_khr_gl_sharing
cl_nv_compiler_options cl_nv_device_attribute_query
cl_nv_pragma_unroll cl_nv_copy_opts
  Platform Extensions function suffix              NV

  Platform Name                                    NVIDIA CUDA
Number of devices                                  1
  Device Name                                       Tesla K80
  Device Vendor                                    NVIDIA Corporation
  Device Vendor ID                                0x10de
  Device Version                                  OpenCL 1.2 CUDA
  Driver Version                                   375.26
  Device OpenCL C Version                         OpenCL C 1.2
  Device Type                                       GPU
  Device Profile                                    FULL_PROFILE
  Device Topology (NV)                             PCI-E, 00:03.6
  Max compute units                                13
  Max clock frequency                              823MHz
  Compute Capability (NV)                          3.7
  Device Partition                                 (core)
    Max number of sub-devices                       1
    Supported partition types                       None
  Max work item dimensions                        3
  Max work item sizes                             1024x1024x64
  Max work group size                             1024
  Preferred work group size multiple              32
  Warp size (NV)                                   32

```

Manual Method

Download  this file and unpack it:

```
curl https://codeload.github.com/hpc12/tools/tar.gz/master | tar
xvfz -
```

Then compile the code:

```
cd tools-master
make
```


If your ICD loader and its header files (CL/cl.h) are in non-standard locations, you can say

```
make OPENCL_INC=/somewhere/include OPENCL_LIB=/somewhere/lib
```

Then run the following:

```
./print-devices
```

On my machine, this prints the following

```
platform 0: vendor 'Intel(R) Corporation'
  device 0: '      Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz'
platform 1: vendor 'Advanced Micro Devices, Inc.'
  device 0: 'Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz'
```

If the output is empty, then no ICDs were found, i.e. something went wrong with the installation.

To run a simple functional test of a device, do the following:

```
./cl-demo 1000000 10
```

You will be prompted for where to run. A sample session might look as follows--make sure it says 'GOOD' at the end:

```
Choose platform:
[0] Intel(R) Corporation
[1] Advanced Micro Devices, Inc.
Enter choice: 0
Choose device:
[0]      Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
Enter choice: 0
-----
-
NAME:      Intel(R) Core(TM) i7-2620M CPU @ 2.70GHz
VENDOR: Intel(R) Corporation
PROFILE: FULL_PROFILE
VERSION: OpenCL 1.1 (Build 31360.31426)
EXTENSIONS: cl_khr_fp64 cl_khr_icd cl_khr_global_int32_base_atomics
cl_khr_global_int32_extended_atomics cl_khr_local_int32_base_atomics
cl_khr_local_int32_extended_atomics cl_khr_byte_addressable_store
cl_intel_printf cl_ext_device_fission cl_intel_exec_by_local_thread
DRIVER_VERSION: 1.1


Type: CPU
EXECUTION_CAPABILITIES: Kernel Native
```

```

GLOBAL_MEM_CACHE_TYPE: Read-Write (2)
CL_DEVICE_LOCAL_MEM_TYPE: Global (2)
SINGLE_FP_CONFIG: 0x7
QUEUE_PROPERTIES: 0x3

VENDOR_ID: 32902
MAX_COMPUTE_UNITS: 4
MAX_WORK_ITEM_DIMENSIONS: 3
MAX_WORK_GROUP_SIZE: 1024
PREFERRED_VECTOR_WIDTH_CHAR: 16
PREFERRED_VECTOR_WIDTH_SHORT: 8
PREFERRED_VECTOR_WIDTH_INT: 4
PREFERRED_VECTOR_WIDTH_LONG: 2
PREFERRED_VECTOR_WIDTH_FLOAT: 4
PREFERRED_VECTOR_WIDTH_DOUBLE: 2
MAX_CLOCK_FREQUENCY: 2700
ADDRESS_BITS: 64
MAX_MEM_ALLOC_SIZE: 2067921920
IMAGE_SUPPORT: 1
MAX_READ_IMAGE_ARGS: 480
MAX_WRITE_IMAGE_ARGS: 480
IMAGE2D_MAX_WIDTH: 8192
IMAGE2D_MAX_HEIGHT: 8192
IMAGE3D_MAX_WIDTH: 2048
IMAGE3D_MAX_HEIGHT: 2048
IMAGE3D_MAX_DEPTH: 2048
MAX_SAMPLERS: 480
MAX_PARAMETER_SIZE: 3840
MEM_BASE_ADDR_ALIGN: 1024
MIN_DATA_TYPE_ALIGN_SIZE: 128
GLOBAL_MEM_CACHELINE_SIZE: 64
GLOBAL_MEM_CACHE_SIZE: 262144
GLOBAL_MEM_SIZE: 8271687680
MAX_CONSTANT_BUFFER_SIZE: 131072
MAX_CONSTANT_ARGS: 480
LOCAL_MEM_SIZE: 32768
ERROR_CORRECTION_SUPPORT: 0
PROFILING_TIMER_RESOLUTION: 1
ENDIAN_LITTLE: 1
AVAILABLE: 1
COMPILER_AVAILABLE: 1
MAX_WORK_GROUP_SIZES: 1024 1024 1024
-----
-
*** Kernel compilation resulted in non-empty log message.
*** Set environment variable CL_HELPER_PRINT_COMPILER_OUTPUT=1 to
see more.
*** NOTE: this may include compiler warnings and other important
messages
*** about your code.
*** Set CL_HELPER_NO_COMPILER_OUTPUT_NAG=1 to disable this message.
0.001980 s
12.119479 GB/s
GOOD

```

The package also contains a simple interface helper (`cl-helper.h` and `cl-helper.c`) for OpenCL. Feel free to use it under the terms of the  MIT license.

OpenCLHowTo (last edited 2017-09-19 02:57:34 by AndreasKloeckner)