AMD ROCm 1.5 Linux driver-stack is out

ROCm is AMD's open source Linux-driver that brings compute to HSA-hardware. It does not provide graphics and therefore focuses on monitor-less applications like machine learning, math, media processing, machine vision, large scale simulations and more.

For those who do not know HSA, the Heterogeneous Software Architecture defines hardware and software such that different processor types (like CPU, GPU, DSP and FPGA) can seamlessly work together and have fine-grained memory sharing. Read more on HSA here.

About ROCm and it's short history

The driver stack has been on Github for more than a year now. Development is done internally, while communication with users is done mostly via Gitlab's issue tracker. ROCm 1.0 was publicly announced on 25 April 2016. After version 1.0, there now have been 6 releases in only one year - the 4 months of waiting time between 1.4 and 1.5 was therefore relatively long. You can certainly say the development is done at a high pace.

ROCm 1.4 was released end of December and besides a long list of fixed bugs, it had the developer preview of OpenCL 2.0 kernel support added. Support for OpenCL was limited to Fiji (R9 Fury series) and Baffin/Ellesmere (Radeon RX 400 series) GPUs, as these have the best HSA support of current GPU offerings.

Currently not all parts of the driver stack is open source, but the binary blobs will be open sourced eventually. You might think why a big corporation like AMD would open source such important part of their offering. This makes totally sense if you understand that their most important customers spend a lot of time on making the drivers and their code work together. By giving access to the code, debugging becomes a lot easier and will reduce development time. This will result in less bugs and a shorter time-to-market for the AMD-version of the software.

The OpenCL language runtime and compiler will be open sourced soon, so AMD offers full OpenCL without any binary blob.

What does ROCm 1.5 bring?

Version 1.5 adds improved support for OpenCL, where 1.4 only gave a developer preview. Both feature-support and performance have been improved. Just like in 1.4 there is support for OpenCL 2.0 kernels and OpenCL 1.2 host-code - the tool clinfo mentions there is even some support of 2.1 kernels, but we haven't fully tested this yet.

The command-line based administration (ROCm-SMI) adds power monitoring, so power-efficiency can be measured.
The HCC compiler was upgraded to the latest CLANG/LLVM. There also have been big improvement in C++ compatibility.

Other improvements:

- Added new API hipHccModuleLaunchKernel which works exactly as hipModuleLaunchKernel but takes OpenCL programming models launch parameters. And its test
- Added new API hipMemPtrGetInfo
- Added new field to hipDeviceProp_t -> gcnArch which returns 803, 700, 900, etc.,

Bug fixes:

- Fixed Copyright and header names
- Fixed issue with bit_extract sample
- Enabled lgamma and lgammaf
- Added guard for GFX8 specific intrinsics
- Fixed few issues with operator overloading of vector data types
- Fixed atanf
- Added guard for __half data types to work with clang version more than 3. (Will be removed eventually).
- Fixed 4_shfl to work only for gfx803 as hawaii don't support permute ops

Current hardware support:

GFX7: Radeon R9 290 4 GB, Radeon R9 290X 8 GB, Radeon R9 390 8 GB, Radeon R9 390X 8 GB, FirePro W9100 (16GB), FirePro S9150 (16 GB), and FirePro S9170 (32 GB).
GFX8: Radeon RX 480, Radeon RX 470, Radeon RX 460, Radeon R9 Nano, Radeon R9 Fury, Radeon R9 Fury X, Radeon Pro WX7100, Radeon Pro WX5100, Radeon Pro WX4100, and FirePro S9300 x2.

If you're buying new hardware, pick a GPU from the GFX8 list. FirePro S9300 X2 is currently the server-grade solution of choice.

Keep an eye on the Phoronix website, which is usually first with benchmarking AMD's open source drivers.

Install ROCm 1.5

Where 1.4 had support for Ubuntu 14.04, Ubuntu 16.04 and Fedora 23, 1.5 added support for Fedora 24 and dropped support for Ubuntu 14.04 and Fedora 23. On other distributions than **Ubuntu 16.04** or **Fedora 24** it *could* work, but there are zero guarantees.

Follow the instructions on Github step-by-step to get it installed via **deb** or **rpm**. Be sure to uninstall any previous release of ROCm to avoid problems.

The part on Grub might not be clear. For this release the magic GRUB_DEFAULT line on Ubuntu 16.04 is:
GRUB_DEFAULT="Advanced options for Ubuntu>Ubuntu, with Linux 4.9.0-kfd-compute-rocm-rel-1.5-76"
You need to alter this line with every update, else it'll keep using the old version.

Make sure "/opt/rocm/bin/" is in your PATH when wanting to do some coding. When running the test, you should get:

```
/opt/rocm/hsa/sample$ sudo make
gcc -c -I/opt/rocm/include -o vector_copy.o vector_copy.c -std=c99
gcc -Wl,--unresolved-symbols=ignore-in-shared-libs vector_copy.o -L/opt/rocm/lib -lhsa-runtime64 -o vector_copy
/opt/rocm/hsa/sample$ ./vector_copy
Initializing the hsa runtime succeeded.
Checking finalizer 1.0 extension support succeeded.
Generating function table for finalizer succeeded.
Getting a gpu agent succeeded.
Querying the agent name succeeded.
The agent name is gfx803.
Querying the agent maximum queue size succeeded.
The maximum queue size is 131072.
Creating the queue succeeded.
"Obtaining machine model" succeeded.
"Getting agent profile" succeeded.
Create the program succeeded.
Adding the brig module to the program succeeded.
Query the agents isa succeeded.
Finalizing the program succeeded.
Destroying the program succeeded.
Create the executable succeeded.
Loading the code object succeeded.
Freeze the executable succeeded.
Extract the symbol from the executable succeeded.
Extracting the symbol from the executable succeeded.
Extracting the kernarg segment size from the executable succeeded.
Extracting the group segment size from the executable succeeded.
Extracting the private segment from the executable succeeded.
Creating a HSA signal succeeded.
Finding a fine grained memory region succeeded.
Allocating argument memory for input parameter succeeded.
Allocating argument memory for output parameter succeeded.
Finding a kernarg memory region succeeded.
Allocating kernel argument memory buffer succeeded.
Dispatching the kernel succeeded.
Passed validation.
Freeing kernel argument memory buffer succeeded.
Destroying the signal succeeded.
Destroying the executable succeeded.
Destroying the code object succeeded.
Destroying the queue succeeded.
Freeing in argument memory buffer succeeded.
Freeing out argument memory buffer succeeded.
Shutting down the runtime succeeded.
```

Also clinfo (installed from the default repo) should work.

Got it installed and tried your code? Did you see improvements? Share your experiences in the comments!

Not really ROCk-music, but this blog has been written while listening to the latest album of the Gorillaz