Subject: Re: CUDA support inside containers
Posted by abufrejoval on Mon, 21 Nov 2016 01:11:11 GMT
View Forum Message <> Reply to Message

I've spent the week-end digging deeper

I think we can safely put aside the installation option disucssion: How you install CUDA inside the container doesn't really seem to matter.

The problem is with the behaviour of the CUDA run-time libraries and I have found no way to influence that anywhere in the documentation.
It's possible that this behaviour changed with the CUDA releases, but I couldn't downgrade on my primary system as that uses a GTX 1070, which requires CUDA 0.8.

The secondary has a GTX 980ti, so there I may be able to go back to CUDA 0.7, should that be necessary for verfication.

As I described I am using the secondary system for baseline comparison and it running a plain CentOS 7 instead.

(Turns out important, as neither LXC nor Docker will run on OpenVZ7, but that's another issue...)

I'm always running the very same samples outside the container on the host and inside the container to compare and using strace as the only means to follow what's going on.

I can see the CUDA runtime going through the /proc and /sys filesystem extensively (/proc/modules is just the first of many files inspected) and once it's happy to find all modules loaded it will open the CUDA devices ('/dev/nvidia-uvm' first), mmap them and follow up with some ioctl(): Obviously I can't see the memory channel interaction with the device.

Inside my Ubuntu based LXC container CUDA applications succeed with all their /procFS searches, are happy with the loaded modules (so they won't try to load them themselves) but then fail to open '/dev/nvidia-uvm' with 'operation not permitted', even if the device files inside the container have world RW permissions. All of that happens as part of the very first CUDA API call ('cudaGetDeviceCount()') and cannot be influenced by compile or startup options.

Looking closer at the LXC configuration I got the impression that UID mapping for non-privileged LXC containers is somewhat inbred into Ubuntu and that compatibility with CentOS is poor (there are UID/GID mapping extensions to 'adduser' RHEL doesn't know about). RHEL depreciated LXC almost immediately after RHEL 7 came out (bastards!) and it's showing (can't decide if they are just lazy or they try to hurt Ubuntu).

I then tried with Docker, not the RHEL supplied one, but the newest 1.12 from Docker, because that's what Nvidia worked with when they created their sample images.

Of course it took me a while to get the storage setup working with that one...

Those two sample Docker images from the Nvidia repository seemed to work: Unfortunately they

are so minimal I couldn't even run an strace inside, so again I cannot really compare against outsie or LXC.

And then they stopped working today, evidently because the Docker image got updated to a slightly newer CUDA runtime than the host...

I'm now setting up an Ubuntu host, to try to get an LXC baseline working and because that seems to be somewhat more popular among the AI research crowd.

However that's not what I want to use going forward: I've been a huge OpenVZ fan for ten years now (bet the company on it, too) and not eager to relearn an Ubuntu userland either.

So I hope I can count on you guys making CUDA work going forward, because currently CUDA is the undisputed king in AI accelleration and that's why its support is a make or break issue.

Ah, yes I did bind mount a couple of files copies from the host /proc to inside the container in an attempt to make the run-time happy, but I had to stop when I couldn't find a way to create (or overlay) the /proc/driver directly: ProcFS won't allow me to create directories.

Somehow I haven't found a way yet to overlay a sparse directory tree where any file or directoy not present in the overlay will be used from the underlying FS. I thought that was supported already, but perhaps that's just wishful thinking (also a bit of a nightmare).