Restrict the access of GPU by enabling an interface via the kernel

Bhavya Agarwal, Dushyant Goyal

01 November, 2013

1 Introduction

The problem was to implement an interface which takes away the functionality of GPU from the user-space codes which in turn has to go through kernel in order to obtain it. In the next section, we show how it was done by shifting one module from cuda-convnet program from accessing GPU directly, to use the interface implemented.

2 Steps of the project

2.1 Kernel Modules

Since we could not find a way to compile the kernel with nvcc in order to implement a system call, we decided to use kernel modules which with the help of ioctl calls, requests the functionality from the GPU.

2.2 Setting up Cuda on our own machine

We tried to set up CUDA on our machine only to come to know that the card NVIDIA GT420M is a hybrid card and it is not possible to install drivers directly. Hence, we had to install and use bumblebee, which is made specially for such cards, in order to pipe and run CUDA commands. However, this created further problems and this was resolved by root access to wildgoose machine with a proper NVIDIA card.

2.3 Exploring User Mode Helper API

Here we use the usermode helper API in order to call the user-space binaries which expose us to the functionality of the GPU. This API's basic documentation is here.

2.4 Understanding "cuda-convnet"

This library is an implementation of convolutional neural-networks over cuda and is available here. We went through the basic structures of code in order to extend it.

2.5 Running RGBToLAB module of cuda-convnet via kernel interface

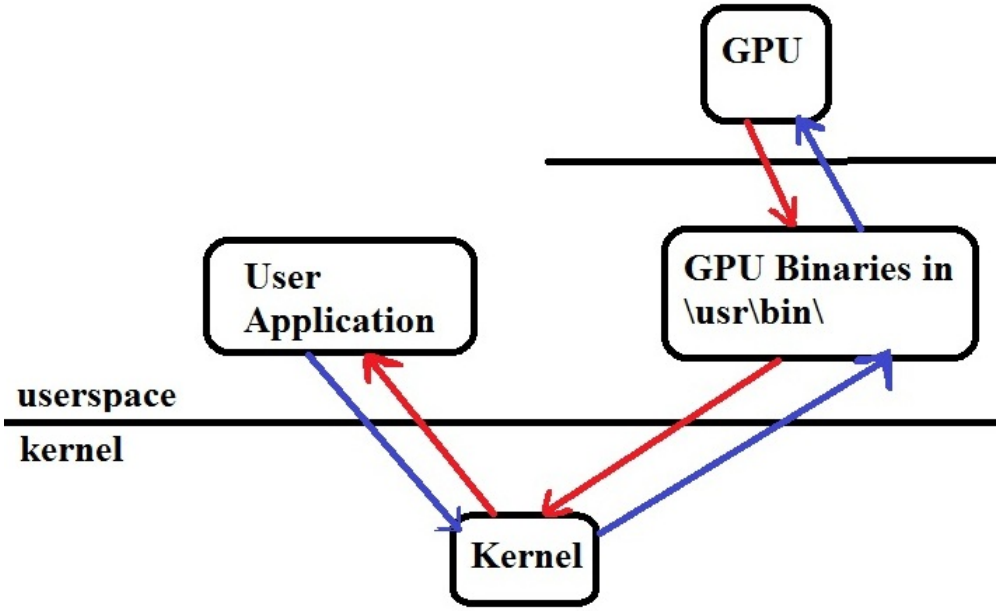
Finally we implemented the required extensions to cuda-convnet, the device driver with required ioctl and the binary for the required purpose.
The cuda-convnet works on layers and execute the layers on the GPU. In our project we have taken a module which converts all the images in the data layer from RGB to LAB color space. LAB is a different color space which is many times a preprocessing for image processing learning based applications. The format of layer parameters is as follows -

[*data*]
type=data
dataIdx=0

The things in square brackets are user-friendly layer names. The type=data line indicates that this is a data layer. dataIdx=0 indicates that the layer named data is mapped to the CIFAR-10 images

Now, we define a new layer "lab" which does RGB2LAB conversion.
[*lab*]
type=rgb2lab
inputs=data
center=true
So, we define two layers: data and lab.

Finally we add the LAB layer to one of the existing Training Layer Parameters - layers-conv-local-11pct.cfg. This layer i.e. RGBtoLAB is executed via the kernel module implemented by us.



3 Running and Validation - Documentation

The changes were

3.1 Compiling

Please note that all the binaries are already in place and these steps are not necessary. They can be executed in following order.
We made a loadable kernel module called 'transform'. For compiling the transform device module, go to cuda-convnet-read-only/lkm and run make. The commands are -
1. $cd cuda-convnet-read-only/lkm
2. $make
If you want to compile the cuda binary, go to cuda-convnet-read-only/lkm and run "nvcc RGBtoLAB.cu -o RGBtoLAB" . Copy the generated binary to '/usr/bin'. The commands are -
1. $nvcc RGBtoLAB.cu -o RGBtoLAB
2. $sudo cp RGBtoLAB /usr/bin
Finally go to the folder cuda-convnet-read-only/ and run 'sh build.sh'. The commands are -
1. cd ../
2. $sh build.sh

3.2 Execute

We were successfully able to implement the kernel module to give access restrict the access of GPU. The following steps need to be followed on the wildgoose inside user bagarwal's home folder -

1. $cd cuda-convnet-read-only/lkm
2. $sudo insmod transform.ko
3. $cd ../
4. $python convnet.py –data-path=../cifar-10-batches-bin/ –save-path=../tmp –test-range=6 –train-range=1-5 –layer-def=./example-layers/layers-conv-local-11pct.cfg –layer-params=./example-layers/layer-params-conv-local-11pct.cfg –data-provider=cifar –test-freq=13

4 Results and Conclusion

Here you can see that the difference is printed between both the cases, i.e. with and without the interface. The difference comes to be around 20,000 for around 400,000 values. The values range from few hundreds to hundreds of thousands. Hence for so many large values, we can conclude that it is due to the difference in floating point operations between the 2 hardwares.

The module which was converted applies LAB transformation for RGB images. This is a linear transformation on all the channels which reduces the effect of lighting.

This module can be extended to increase an extra later of security for the GPU access. We also did an implementation of same using opencv library binaries and obtained decent results. However, in the given case due to excess of file operations, this interface made the implementation 30 times slow.