CSC Deutschland GmbH

Get ready for GPU Computing with ThinkPad W-Series

Ubuntu 14.04 **+ CUDA** 7.5 **+ cuDNN** 4
Anaconda 2-2.5.0 **+ TensorFlow** 0.7.1

Autor: Zilong Zhao
🖂: zzhao3@csc.com
March 22, 2016

Contents

This document was created with LaTeX, a typesetting system for
What you think is what you get.

Version 1

1 Introduction

GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications. Pioneered in 2007 by NVIDIA, GPU accelerators now power energy-efficient datacenters in government labs, universities, enterprises, and small-and-medium businesses around the world. GPU-accelerated computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster. A simple way to

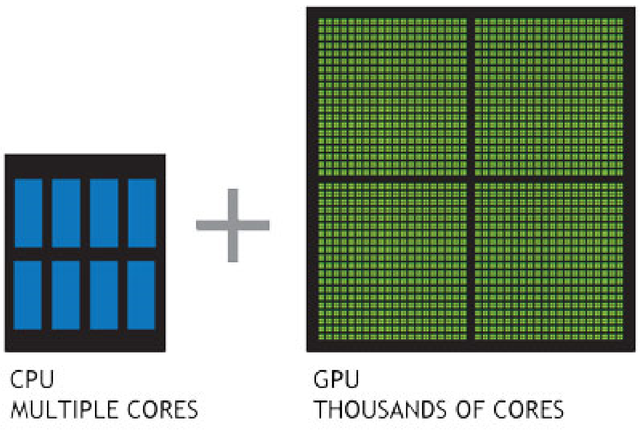


Figure 1: CPU vs. GPU[1]

understand the difference between a CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.[1]

The aim of this tutorial is to setup the `ThinkPad W541` from CSC and get it ready for GPU computing. `ThinkPad W541` has a graphics chip `NVIDIA Quadro K1100M 2G`, which has the Compute Capability 3.0[2].

[1]See [1]
[2]Source: `https://developer.nvidia.com/cuda-gpus`

2 Installation

GPU-accelerated computing requires the parallel computing platform CUDA as well as the TensorFlow framework on a running Python environment (e.g. Anaconda on Linux). Therefore, before we start the Installation process, Please use the following checklist to ensure that all requirements are fulfilled.

- A CUDA supported GPU (e.g. `NVIDIA Quadro K1100M` for this tutorial),
- Ubuntu 14.04, 64-bit[1],
- CUDA 7.5[2],
- cuDNN 4[3],
- Anaconda 2-2.5.0[4],
- TensorFlow 0.7.1[5].

Unfortunately the virtual machine simulates a graphics device and as such we won't have access to the real GPU. This is because of the way the virtualisation handles multiple VMs accessing the same device. So we're out of luck with the virtualisation route and limited to the following:

1. Unrealistic: Format the internal hard drive and install Linux instead,
2. Install Linux alongside in an external USB 3.0 hard disk (the best option).

Due to the Microsoft based working platform in CSC, the second choice should be a reasonable solution. Suppose we've successfully installed the Ubuntu 14.04, 64-bit on the external hard disk. Let's initialize and prepare for the GPU computing.

[1]Source: `http://www.ubuntu.com/download/desktop`
[2]Source: `https://developer.nvidia.com/cuda-downloads`
[3]Source: `https://developer.nvidia.com/cudnn`
[4]Source: `http://repo.continuum.io/archive/Anaconda2-2.5.0-Linux-x86_64.sh`
[5]Source: `https://www.tensorflow.org/`

2.1 CUDA installation

The first step is to verify that the GPU is CUDA-capable, from the command line, enter:

```
$ lspci | grep -i nvidia
```

If graphics card is from NVIDIA and it is listed in `http://developer.nvidia.com/cuda-gpus`, means the GPU is CUDA-capable.

The gcc compiler is required for development using the CUDA Toolkit. It is not required for running CUDA applications. It is generally installed as part of the Linux installation, and in most cases the version of gcc installed with a supported version of Linux will work correctly. To verify the version of gcc installed on your system, type the following on the command line:

```
$ gcc --version
```

The NVIDIA CUDA Toolkit is available at `http://developer.nvidia.com/cuda-downloads`. Choose

Linux > x86_64 > Ubuntu > 14.04 > deb (local),

the 1.9GB `.deb` package `cuda-repo-ubuntu1404-7-5-local_7.5-18_amd64.deb` will be downloaded. Go to the downloads directory, and follow the installation instructions:

```
$ sudo dpkg -i
↪  cuda-repo-ubuntu1404-7-5-local\_7.5-18\_amd64.deb
$ sudo apt-get update
$ sudo apt-get install cuda
```

After that, the post-installation actions must be manually performed, i.e. change the environment variables for 64-bit operating systems. In Ubuntu, we edit the `~/.bashrc` file by adding

```
export PATH=/usr/local/cuda-7.5/bin:$PATH
export
  ↪  LD_LIBRARY_PATH=/usr/local/cuda-7.5/lib64:$LD_LIBRARY_PATH
```

Log out and log in again to activate the new variables (close and open the terminal).

2.2 Check CUDA installation

To verify the integrity of the installation. A convenience installation script is provided, install the samples in a directory `cuda-samples` under home:

```
$ cuda-install-samples-7.5.sh ~/cuda-samples
```

Then we can check the version of the NVIDIA drivers by executing the command

```
$ cat /proc/driver/nvidia/version
```

The version of the CUDA Toolkit can be checked by running

```
$ nvcc -V
```

in a terminal window. The NVIDIA CUDA Toolkit includes sample programs in source form. We should compile them, to do this, go to the samples directory:

```
cd ~/cuda-samples/NVIDIA_CUDA-7.5_Samples
```

and build the samples:

```
make -j $(($(nproc) + 1))
```

Note: you could just run `make`, but that last part `(-j $(($(nproc) + 1)))` executes the `make` command in parallel, using the number of cores in your machine (plus one), so the compilation would finish faster. Then restart the computer.

Now we Check the CUDA installation by running

```
~/NVIDIA_CUDA-7.5_Samples/bin/x86_64/linux/release/deviceQuery
```

Check that it detects a CUDA device, check that it detects the graphics card, check at the end that it says the test passed. Run the bandwidth test to check that the system and the CUDA device are capable to communicate:

```
~/NVIDIA_CUDA-7.5_Samples/bin/x86_64/linux/release/bandwidthTest
```

Note that the measurements for your CUDA-capable device description will vary from system to system. The important point is that you obtain measurements, and that the second-to-last line confirms that all necessary tests passed. Should the tests not pass, make sure you have a CUDA-capable NVIDIA GPU on your system and make sure it is properly installed.

2.3 Install OpenBLAS

Install Fortran compiler:

```
sudo apt-get install gfortran
```

Create a `code` directory to store your code:

Note: you can put your code wherever you want, so, modify this part as you wish, but I will assume that your are doing it.

```
mkdir ~/code
```

Enter that directory:

```
cd ~/code
```

Clone OpenBLAS:

```
git clone https://github.com/xianyi/OpenBLAS.git
```

Go to the OpenBLAS directory:

```
cd OpenBLAS
```

Compile OpenBLAS:

```
make -j $(($(nproc) + 1))
```

Install OpenBLAS:

```
sudo make PREFIX=/usr/local install
```

2.4 Install Boost

```
sudo apt-get install libboost-all-dev
```

2.5 Install OpenCV

```
sudo apt-get install libopencv-dev
```

2.6 Install: protobuf, glog, gflags

```
sudo apt-get install libprotobuf-dev libgoogle-glog-dev
  ↪  libgflags-dev protobuf-compiler
```

2.7 Install IO libraries: hdf5, leveldb, snappy, lmdb

```
sudo apt-get install libhdf5-serial-dev libleveldb-dev
  ↪  libsnappy-dev liblmdb-dev
```

2.8 Install Anaconda Python

Download Anaconda 2-2.5.0 for Linux: `http://repo.continuum.io/archive/Anaconda2-2.5.0-Linux-x86_64.sh`

Go to the downloads directory:

```
cd ~/Downloads
```

Execute the installer:

```
bash Anaconda2-2.5.0-Linux-x86_64.sh
```

When asked if add Anaconda to the PATH for the current user, type `yes`. Log out and log in again to activate the new variables (close and open the terminal).

2.9 Install HDF5

Install HDF5 with conda, although it will only be used by Anaconda:

```
conda install hdf5
```

2.10 Configure the HDF5 version

Note: this is a quick fix for a minor bug / issue with the version of libhdf5. If you know a better / proper way to solve it, let me know. If this section doesn't apply to you, omit it.

Go to the libraries directory:

```
cd /usr/lib/x86_64-linux-gnu
```

Link the system version of HDF5 from 7 to 9:

```
sudo ln -s libhdf5.so.7 libhdf5.so.9
sudo ln -s libhdf5_hl.so.7 libhdf5_hl.so.9
```

Update the "Dynamic Linker":

```
sudo ldconfig
```

2.11 Install CuDNN 4

The NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks. Deep learning developers and researchers worldwide rely on the highly optimized routines in cuDNN which allow them to focus on designing and training neural network models rather than spending time on low-level performance tuning. For more information, please visit `https://developer.nvidia.com/cudnn`

Note: for this to work, we need a NVIDIA GPU based on the **Kepler** architeture, such as the `NVIDIA Quadro K1100M 2G`, or the newer. For downloading the cuDNN, we need to have an account or register in the NVIDIA developer program, it's free: `https://developer.nvidia.com/user`, using the account for downloading: `https://developer.nvidia.com/cuDNN`.

Go to the downloads directory:

```
cd ~/Downloads/
```

Uncompress the CuDNN download:

```
tar xvf cudnn*.tgz
```

Enter the uncompressed directory:

```
cd cuda
```

Copy the `*.h` files to CUDA installation:

```
sudo cp */*.h /usr/local/cuda-7.5/include/
```

Copy the `.so` files to your CUDA installation:

```
sudo cp */*.so* /usr/local/cuda-7.5/lib64/
```

2.12 Install TensorFlow 0.7.1

TensorFlow is an open source software library for machine learning in various kinds of perceptual and language understanding tasks. It is a second-generation API which is currently used for both research and production by different teams in dozens of commercial Google products, such as speech recognition, Gmail, Google Photos, and Search. These teams had previously used DistBelief, a first-generation API. TensorFlow was originally developed by the Google Brain team for Google's research and production purposes and later released under the Apache 2.0 open source license on November 9, 2015.

For Anaconda distruibution of Python, the easiest way to install TensorFlow is with `conda` by running:

```
conda install -c https://conda.anaconda.org/jjhelmus tensorflow
```

Unfortunately, the installation with `conda` doesn't support GPU.

Meanwhile, the installation of GPU enabled version with `pip install` requires the

$$\text{CUDA compute capability} \geq 3.5,$$

but the `NVIDIA Quadro K1100M` graphic chip only supports 3.0[1]. We will encounter this kind of error message

```
tensorflow/core/common_Ignoring gpu device
(device: 0, name: NVIDIA QUADRO K1100M, pci bus
id: 0000:01:00.0) with Cuda compute capability
3.0. The minimum required Cuda capability is 3.5.
```

So we need to build from source. We will explain how to setup Tensorflow with CUDA compute capability 3.0 devices.

Install Bazel

First, we need to install Bazel. I recommend to use Bazel $0.2.0$.

Setup Java JDK-8 Bazel requires java jdk-7 or 8. OpenJDK 8 is not available on Ubuntu 14.04. Follow the steps to install it:

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
```

Install other dependencies

```
$ sudo apt-get install pkg-config zip g++ zlib1g-dev unzip
```

Download Bazel 0.2.0 at `https://github.com/bazelbuild/bazel/releases/download/0.2.0/bazel-0.2.0-installer-linux-x86_64.sh`

Compile Bazel by running the installer

[1]https://developer.nvidia.com/cuda-gpus

```
$ chmod +x bazel-version-installer-os.sh
$ ./bazel-version-installer-os.sh --user
```

After the installation, Bazel executable is located at:

```
$HOME/bin/bazel
```

Finally, we start to

Install TensorFlow

Install other dependencies

```
$ sudo apt-get install swig wheel git
```

Download Tensorflow 0.7.1

```
$ git clone https://github.com/tensorflow/tensorflow.git
```

Compile TensorFlow Configurate TensorFlow built with cuda 3.0 compute capability

```
$ cd tensorflow
$ TF_UNOFFICIAL_SETTING=1 ./configure
```

```
Please specify the location of python. [Default is ...]:
Do you wish to build TensorFlow with GPU support? [y/N] y
GPU support will be enabled for TensorFlow

Please specify the Cuda SDK version you want to use, e.g. 7.0.
   ↪  [Leave
empty to use system default]: 7.5

Please specify the location where CUDA 7.5 toolkit is
   ↪  installed. Refer to
README.md for more details. [default is: /usr/local/cuda-7.5]:
   ↪  /usr/local/cuda-7.5

Please specify the Cudnn version you want to use. [Leave empty
   ↪  to use system
default]: 4.0.4

Please specify the location where the cuDNN 4.0.4 library is
   ↪  installed. Refer to
README.md for more details. [default is: /usr/local/cuda-7.5]:
   ↪  /usr/local/cuda-7.5/

Please specify a list of comma-separated Cuda compute
   ↪  capabilities you want to
build with. You can find the compute capability of your device
   ↪  at:
https://developer.nvidia.com/cuda-gpus.
Please note that each additional compute capability
   ↪  significantly increases your
build time and binary size. [Default is: \"3.5,5.2\"]: 3.0

Setting up Cuda include
Setting up Cuda lib64
Setting up Cuda bin
Setting up Cuda nvvm
Configuration finished
```

Before we compile tensorflow, we need to run

```
git submodule update --init
```

otherwise, it will raise some error by compiling tensorflow as following

```
.......
ERROR:
  ↪  /share/sw/admin/sources/t/tensorflow/tensorflow/cc/BUILD:63:1:
  ↪  error loading package 'tensorflow/core': Extension file
  ↪  not found. Unable to load package for
  ↪  '//google/protobuf:protobuf.bzl': BUILD file not found on
  ↪  package path and referenced by
  ↪  '//tensorflow/cc:tutorials_example_trainer'.
ERROR: Loading failed; build aborted.
INFO: Elapsed time: 1.988s
```

Build the target with GPU support

```
$ $HOME/bin/bazel build -c opt --config=cuda
  ↪  //tensorflow/cc:tutorials_example_trainer
$ bazel-bin/tensorflow/cc/tutorials_example_trainer --use_gpu
```

After compiling, you should get a output like this:

```
# Lots of output. This tutorial iteratively calculates the
  ↪  major eigenvalue of
# a 2x2 matrix, on GPU. The last few lines look like this.
000009/000005 lambda = 2.000000 x = [0.894427 -0.447214] y =
  ↪  [1.788854 -0.894427]
000006/000001 lambda = 2.000000 x = [0.894427 -0.447214] y =
  ↪  [1.788854 -0.894427]
000009/000009 lambda = 2.000000 x = [0.894427 -0.447214] y =
  ↪  [1.788854 -0.894427]
```

Because we will use TensorFlow python interface, we need to create a pip package to install it.

Create pip package First step is to use bazel to create pip package with GPU support

```
$ $HOME/bin/bazel build -c opt --config=cuda
  ↪  //tensorflow/tools/pip_package:build_pip_package
```

Finally, we move to the last step of installation:

Install python tensorflow We use the pip package that we created to install tensorflow in python

```
$ bazel-bin/tensorflow/tools/pip_package/build_pip_package
  ↪ /tmp/tensorflow_pkg

# The name of the .whl file will depend on your platform.
$ pip install
  ↪ /tmp/tensorflow_pkg/tensorflow-0.7.1-py2-none-linux_x86_64.whl
```

Now it's the time to test if everything is working well.

3 Testing

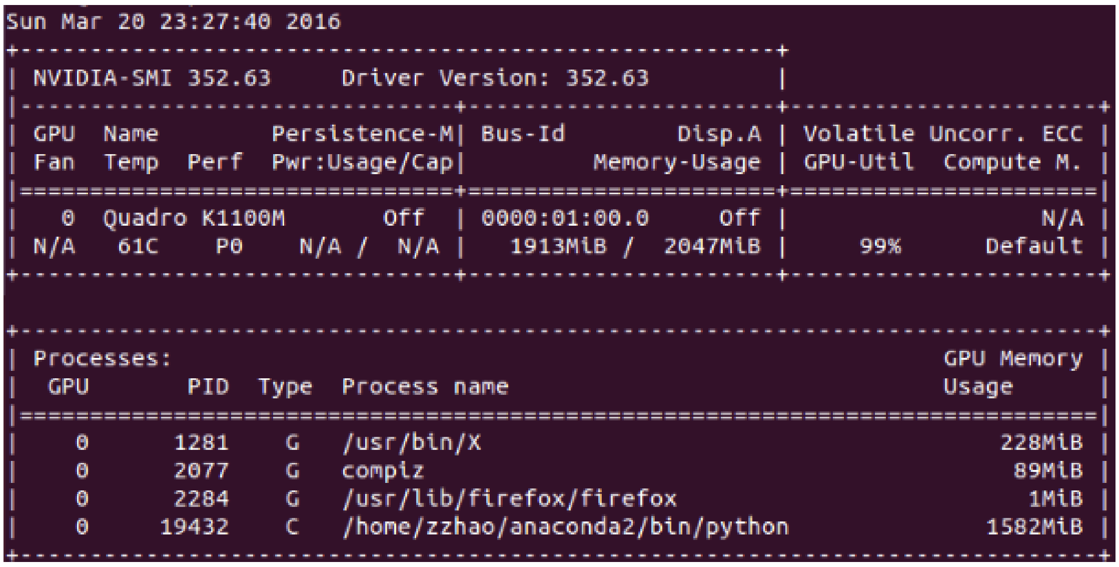
Train our first TensorFlow neural network model

```
$ cd tensorflow/models/image/mnist
$ python convolutional.py
Successfully downloaded train-images-idx3-ubyte.gz 9912422
  ↪ bytes.
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877
  ↪ bytes.
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting data/train-images-idx3-ubyte.gz
Extracting data/train-labels-idx1-ubyte.gz
Extracting data/t10k-images-idx3-ubyte.gz
Extracting data/t10k-labels-idx1-ubyte.gz
Initialized!
Epoch 0.00
Minibatch loss: 12.054, learning rate: 0.010000
Minibatch error: 90.6%
Validation error: 84.6%
Epoch 0.12
Minibatch loss: 3.285, learning rate: 0.010000
Minibatch error: 6.2%
Validation error: 7.0%
...
...
```

Meanwhile, in a new terminal, by typing

```
nvidia-smi
```

you will see



```
Sun Mar 20 23:27:40 2016
+------------------------------------------------------+
| NVIDIA-SMI 352.63     Driver Version: 352.63         |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Quadro K1100M       Off  | 0000:01:00.0     Off |                  N/A |
| N/A   61C    P0    N/A /  N/A |   1913MiB /  2047MiB |     99%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID  Type  Process name                               Usage      |
|=============================================================================|
|    0      1281    G   /usr/bin/X                                   228MiB |
|    0      2077    G   compiz                                        89MiB |
|    0      2284    G   /usr/lib/firefox/firefox                       1MiB |
|    0     19432    C   /home/zzhao/anaconda2/bin/python            1582MiB |
+-----------------------------------------------------------------------------+
```

Figure 2: Screenshot of nvidia-smi[1]

the `PID 19432` in Fig. 2 shows that GPU and GPU memory is being used for computing, and the GPU computing should work well. Congratulations! Enjoy your new GPU-accelerated computing era.

If you meet any difficulty during the installation, feel free to drop a mail!

Bibliography

[1] www.nvidia.com *WHAT IS GPU ACCELERATED COMPUTING?* http://www.nvidia.com/object/what-is-gpu-computing.html

[2] https://developer.nvidia.com/ *CUDA GPUs* https://developer.nvidia.com/cuda-gpus

[3] http://docs.nvidia.com *NVIDIA CUDA Getting Started Guide for Linux* http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/

[4] https://github.com/tiangolo *Installation: Ubuntu (step by step, including external libraries)* https://github.com/tiangolo/caffe/blob/ubuntu-tutorial-b/docs/install_apt2.md

[5] http://spturtle.blogspot.de *CUDA7.5 and Theano Setup in Ubuntu linux for Deep learning* http://spturtle.blogspot.de/2015/07/cuda70-and-theano-setup-in-ubuntu-linux.html

[6] Google TensorFlow *Download and Setup* https://www.tensorflow.org/versions/r0.7/get_started/os_setup.html

[7] http://bazel.io/ *Installing Bazel* http://bazel.io/docs/install.html

[8] www.aymeric.net *Install Tensorflow with cuda 3.0 compute compatibility devices* http://www.aymeric.net/journal/2016/01/install-tensorflow-with-cuda-3-0-compute-compatibility-dev

[9] CTAN *The Comprehensive TEX Archive Network* https://www.ctan.org/

[10] LaTeX Symbol *The Comprehensive LaTeX Symbol List* `http://tug.ctan.org/info/symbols/comprehensive/symbols-a4.pdf`

[11] Wikipedia *Wikipedia, The Free Encyclopedia* `https://en.wikipedia.org/`

[12] GitHub, Inc. (US) *GitHub* `https://github.com/`