BUILD YOUR OWN HIGH PERFORMANCE CLUSTER (WITH CUDA 10.0) STEP BY STEP

A Study Note by ACTION lab, Depts of CEE at Mississippi State University

Dr.Pengfei(Taylor) Li   Peirong(Slade) Wang

Last updated in Nov 7th | 2018

Platform: Ubuntu 18.04
openmpi version 3.1.3
In our example we use 192.168.2.1 and 192.168.2.2 as main node and computing node Ip address.
Lines in Red Color is what you need to type in Machine#1(Main_Node)
Lines in Green Color is what you need to type in Machine#2(Slave_Node)
Lines in Orange Color is what you need to type in both machines.

1. Setting up NFS (Network File System) server on Machine#1(Main_Node)

On Ubuntu-mainnode we`ll set the machine as the NFS server. We will need to install a couple of NFS libraries.

(http://quidsup.net/tutorials/?p=nfs)

(install NFS server)

```
sudo apt-get install nfs-kernel-server
```

(Create Folder List to Share)

```
sudo mkdir /nfsshare (for example)
```

(edit export file to give location and files to share)

```
sudo nano /etc/exports
```

type in

```
/nfsshare    192.168.2.2 (rw,sync,no_root_squash,no_subtree_check)
```

(folder you want to share)   (ip address you want to share to)

`exportfs –a`   (load the/etc/exports new changes)

(start nfs server)

```
sudo service nfs-kernel-server start
```

2. Setting up NFS Client: Machine #2(slave_node)

1. install nfs service and libraries

```
sudo apt-get install nfs-common (install nfs client)
```

2. make a folder where the shared folder from Machine #1 will be mounted on Machine #2.

```
sudo mkdir /nfsshare
```

3. make sure that we can access Ubuntu-mainnode(Machine #1), the NFS Server. Make sure that the following two commands do not return any errors.

```
    showmount -e 192.168.2.1
    rpcinfo -p 192.168.2.1
    mount 192.168.2.1:/nfsshare  /nfsshare
   df -h
```

With df -h, we should see that 10.0.1.2:/nfs mount has been created at the bottom. If we create any file inside /nfs, then all the machines connected can see the same file.

Now, we test that the shared folder actually works.

```
cd /nfsshare
sudo nano 123
```

(create a txt file name 123)

On machine #1 (mainnode), if we cd /nfsshare, we will see 123.txt is inside the folder.

3.Making NFS more automatic

When you restart the two virtual machines, the NFS shared folder will not be there. We need to set a more automatic way for the NFS client to look for the NFS folder.

On the slave_node, we change a file called /etc/fstab.

```
sudo nano /etc/fstab
```

We add the following line:

```
192.168.2.1:/nfsshare /nfsshare  nfs auto,noatime,nolock,bg,nfsvers=3,intr,tcp,actimeo=1800 0 0
```

Every time, we restart the client, we can re-mount the NFS shared folder by typing mount -a.

```
mount -a
```

2. Setting up SSH Keys

Install ssh on both machines

```
sudo apt-get install ssh
```

generate ssh keys

```
ssh-keygen  -t rsa  -b 4096  -C your_email@example.com
```

You can press Enter to leave the next three prompts as default.

Enter file in which to save the key (/Users/*you*/.ssh/id_rsa): *[Press enter]*
Enter passphrase (empty for no passphrase): *[Type a passphrase]*
Enter same passphrase again: *[Type passphrase again]*

```
Your identification has been saved in /Users/you/.ssh/id_rsa.


Your public key has been saved in /Users/you/.ssh/id_rsa.pub.
```

```
The key fingerprint is:


01:0f:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db your_email@example.com
```

Open the ssh folder

```
cd ~/.ssh
```

copy the public key, id_rsa.pub, to authorized_keys to enable this key for access to machine #1

```
cp  id_rsa.pub authorized_keys
```

Now, we should send the private key, id_rsa, and public key, id_rsa.pub, from machine #1 to machine #2. We use a command called scp for copying files over machines.

```
cd ~/.ssh
```

```
scp id_rsa id_rsa.pub username@192.168.2.2:
```

On machine #2, we have received the private key and public key.

Now, we copy the id_rsa and id_rsa.pub to the ~/.ssh folder.

```
sudo cp id_rsa id_rsa.pub  ~/.ssh
```

We want to copy id_rsa.pub to the authorized_keys to allow machine #1 to be able to SSH to machine #2 without a password.(on machine #2)

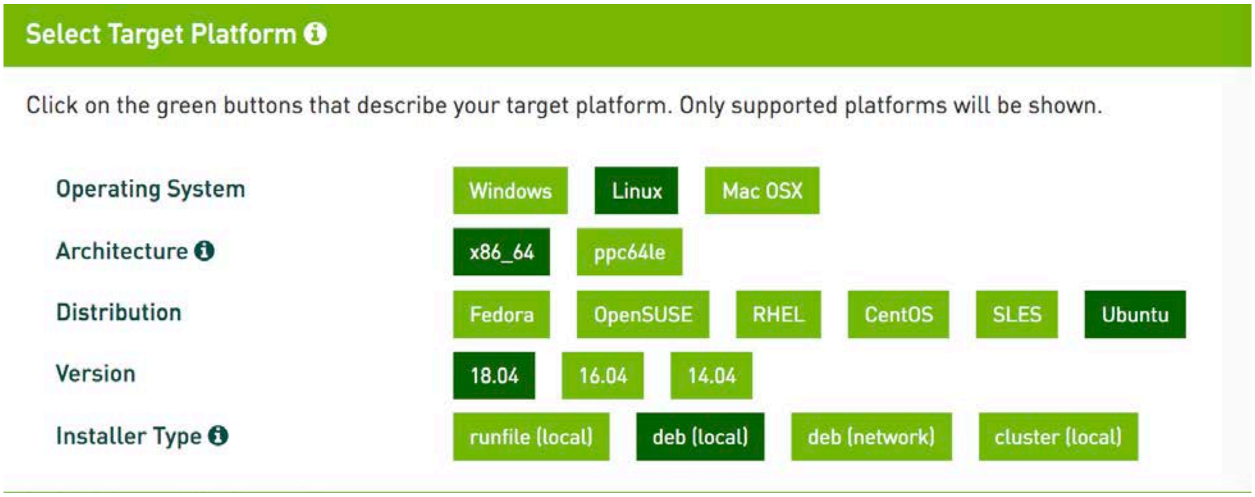
```
cd ~/.ssh
cp id_rsa.pub authorized_keys
```

We should be able to ssh from machine #1 to machine #2 without a password and vice versa.

On machine #1: ssh username@192.168.2.2
On machine #2: ssh username@192.168.2.1

3. Installing CUDA 10.0 (If GPU Computing is needed in Openmpi, if not you can skip this part)

1. Download CUDA 10.0
go to (https://developer.nvidia.com/cuda-downloads)



Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System: Windows [✓ Linux] Mac OSX

Architecture: [✓ x86_64] ppc64le

Distribution: Fedora OpenSUSE RHEL CentOS SLES [✓ Ubuntu]

Version: [✓ 18.04] 16.04 14.04

Installer Type: runfile (local) [✓ deb (local)] deb (network) cluster (local)

3. Move download CUDA file to /nfsshare folder

go to the folder you download CUDA and move CUDA installation file to share folder

```
sudo mv cuda-repo-ubuntu1804-10-0-local-10.0.130-410.48_1.0-1_amd64.deb /nfsshare
```

(Before install CUDA to /nfsshare folder change /nfsshare previllige from root to username)

```
sudo chown –R username /nfsshare
```

install CUDA 10.0 on both machines (one by one)

```
sudo dpkg -i cuda-repo-ubuntu1804-10-0-local-10.0.130-410.48_1.0-1_amd64.deb
sudo apt-key add /var/cuda-repo-<version>/7fa2af80.pub
sudo apt-get update
sudo apt-get install cuda
```

edit ~/.bashrc file (add following two line at the bottom of the bashrc file)

```
nano ~/.bashrc
export PATH=/usr/local/cuda-10.0/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64:$LD_LIBRARY_PATH
```

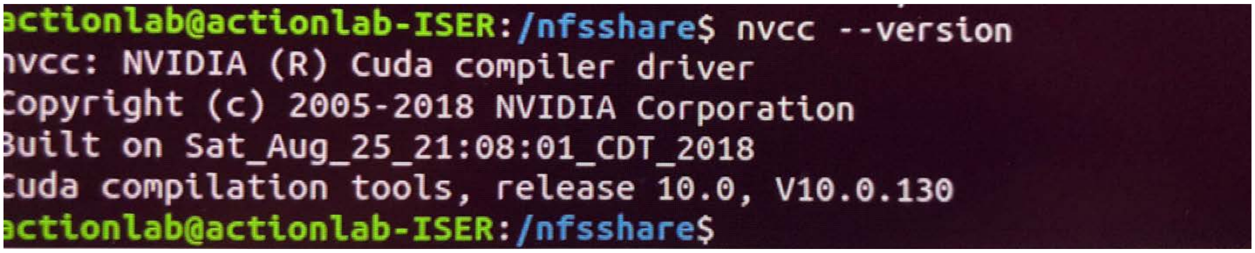
reload bashrc file

```
source ~/.bashrc
```

Reboot both machines
After reboot check if Cuda compiler is successfully installed

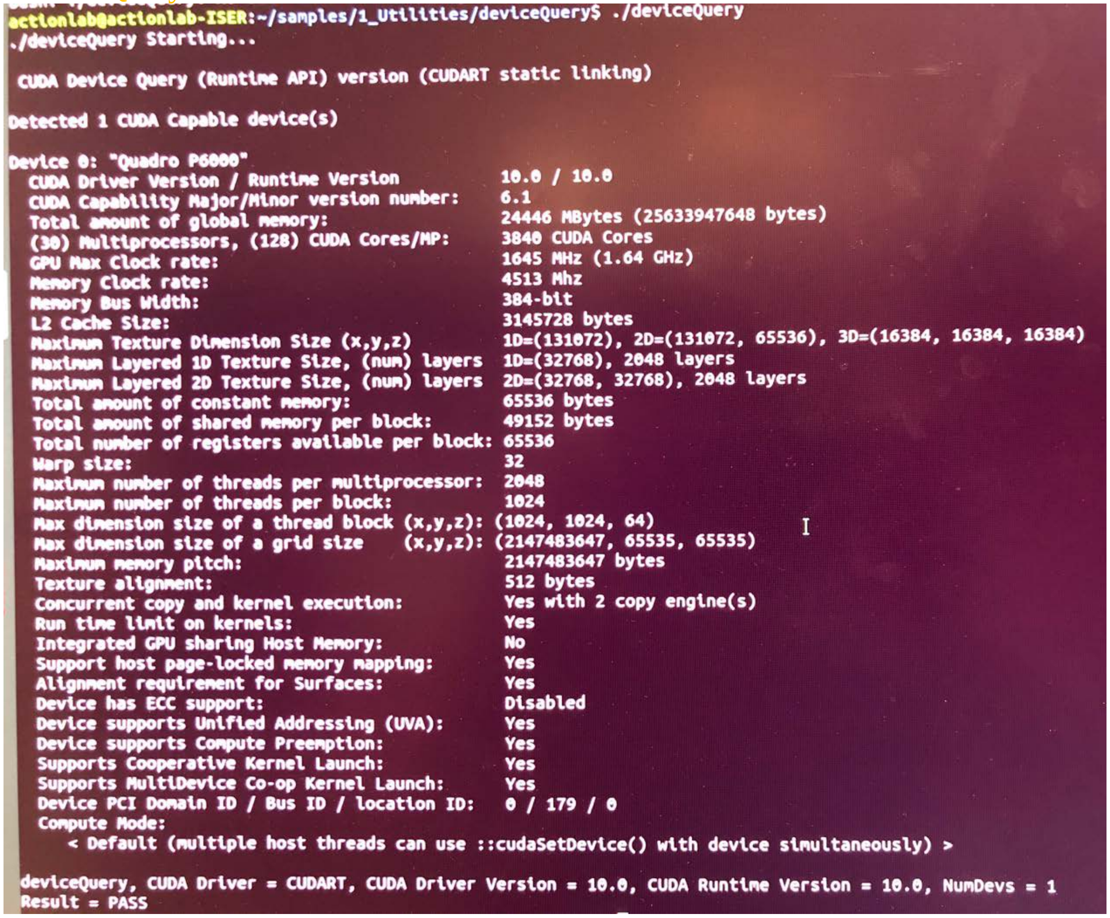
```
nvcc --version
```

Which will give the results of your CUDA toolkit version and built date.



```
actionlab@actionlab-ISER:/nfsshare$ nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Sat_Aug_25_21:08:01_CDT_2018
Cuda compilation tools, release 10.0, V10.0.130
actionlab@actionlab-ISER:/nfsshare$
```

next we will test a sample included in CUDA 10.0 build
go to usr/local/cuda-10.0
copy sample folder to nfsshare
go to 1_Utilities/deviceQuery
right click open terminal here
make
after make
run the sample which will give details of your NIVIDIA GRAPHIC CARD
./deviceQuery



```
actionlab@actionlab-ISER:~/samples/1_Utilities/deviceQuery$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "Quadro P6000"
  CUDA Driver Version / Runtime Version          10.0 / 10.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 24446 MBytes (25633947648 bytes)
  (30) Multiprocessors, (128) CUDA Cores/MP:     3840 CUDA Cores
  GPU Max Clock rate:                            1645 MHz (1.64 GHz)
  Memory Clock rate:                             4513 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 3145728 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 179 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.0, CUDA Runtime Version = 10.0, NumDevs = 1
Result = PASS
```

4. Installing Openmpi-3.1.1

Download openmpi-3.1.1.tar.gz to nfsshare folder

(https://www.open-mpi.org/software/ompi/v3.1/)

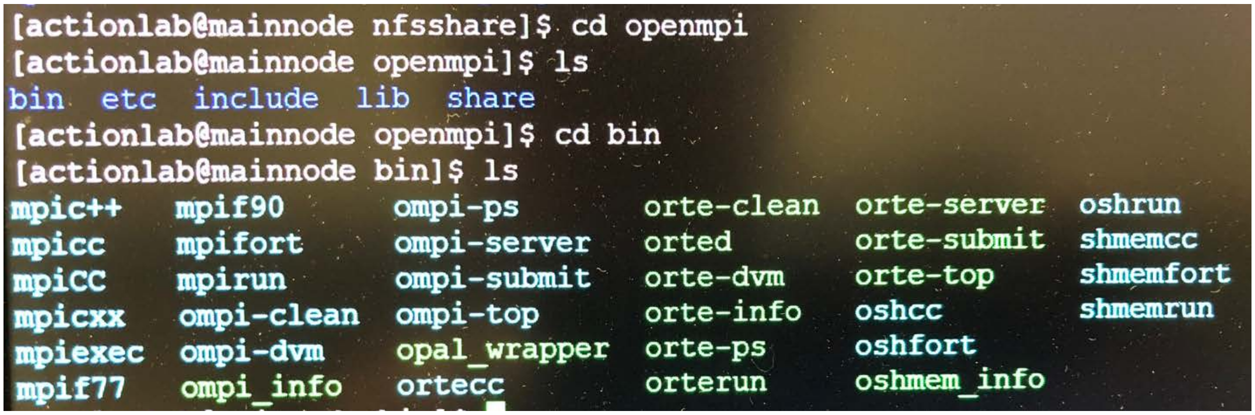
extract the openmpi-3.1.3.tar.gz folder
tar -xvf openmpi-3.1.3.tar.gz

We will make a directory where all the compiled binaries and libraries of openmpi will go.
mkdir /nfsshare/openmpi

configure the settings of openmpi for installation.
cd /nfsshare/openmpi-3.1.3
./configure --prefix=/nfsshare/openmpi --with-cuda (if you follow step 3 installed CUDA)
./configure --prefix=/nfsshare/openmpi (If you did not install CUDA)

Install openmpi-3.1.3
After configure
make
After make
make install

If we cd /nfsshare/openmpi, we will see folders containing the binaries and libraries of openmpi. If we cd /nfsshare/openmpi/bin, we can see openmpi binaries like mpicc and mpirun.



```
[actionlab@mainnode nfsshare]$ cd openmpi
[actionlab@mainnode openmpi]$ ls
bin  etc  include  lib  share
[actionlab@mainnode openmpi]$ cd bin
[actionlab@mainnode bin]$ ls
mpic++    mpif90       ompi-ps       orte-clean   orte-server   oshrun
mpicc     mpifort      ompi-server   orted        orte-submit   shmemcc
mpiCC     mpirun       ompi-submit   orte-dvm     orte-top      shmemfort
mpicxx    ompi-clean   ompi-top      orte-info    oshcc         shmemrun
mpiexec   ompi-dvm     opal_wrapper  orte-ps      oshfort
mpif77    ompi_info    ortecc        orterun      oshmem_info
```

Currently, we won't be able to use mpicc from anywhere on the machine. We need to change the ~/.bashrc file on machine #1 and machine #2 to globalize the mpi commands.

On both machines:

vi ~/.bashrc

At the bottom of ~/.bashrc, add the following two lines:

export PATH=/nfsshare/openmpi/bin:$PATH

export LD_LIBRARY_PATH="/nfsshare/openmpi/lib:$LD_LIBRARY_PATH"

PATH is used for bin folders, and LD_LIBRARY_PATH is used for lib folders. To reload the ~/.bashrc, type the following command on both machines:

```
source ~/.bashrc
```

5. **Using MPI binaries: Running MPI**

Go to the nfsshare folder

```
cd /nfsshare
```

create a folder for projects

```
mkdir /projects
```

create a host file contains IP address for all the IP`s that we want MPI run

```
nano hosts
```

(for example our machine Mainnode has 6 cores and Slave nodes has 40 cores)
add following two line into hosts file you just created

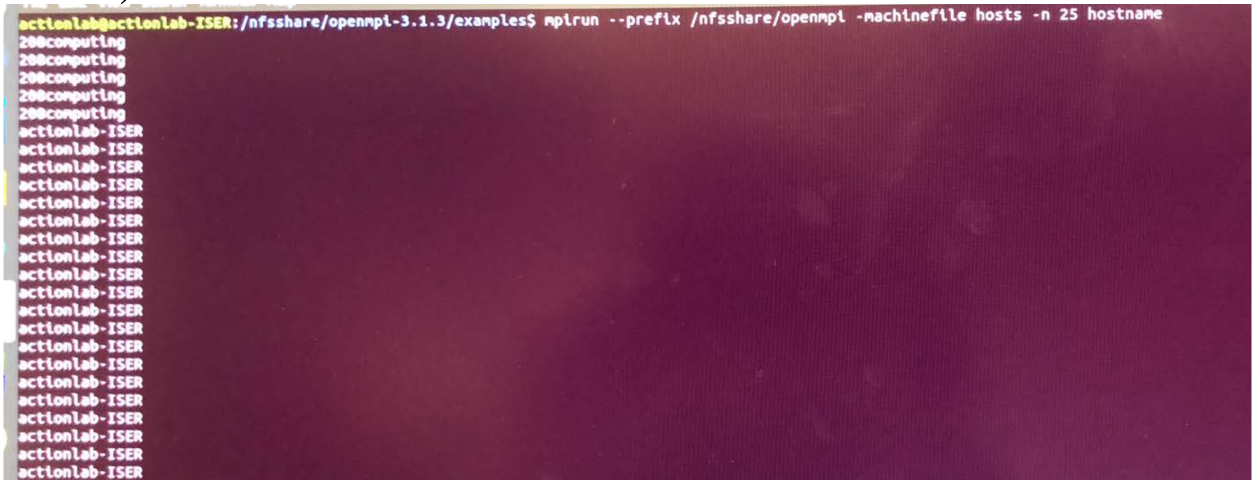
```
192.168.2.1 slots=40


192.168.2.2 slots=6
```

test your openmpi;

```
mpirun --prefix /nfsshare/openmpi -machinefile hosts -n 25 hostname
```

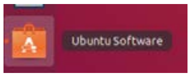
(which will give you the name of the cores mpi is currently using, from picture below you can clearly see five cores of twenty-five are come from 208computing which is the system name of our slave)

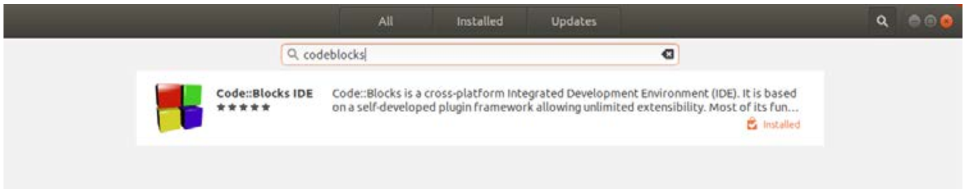


```
actionlab@actionlab-ISER:/nfsshare/openmpi-3.1.3/examples$ mpirun --prefix /nfsshare/openmpi -machinefile hosts -n 25 hostname
208computing
208computing
208computing
208computing
208computing
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
actionlab-ISER
```

6.Install Codeblocks and anaconda

in Ubuntu codeblocks can be directly download and installed in software market





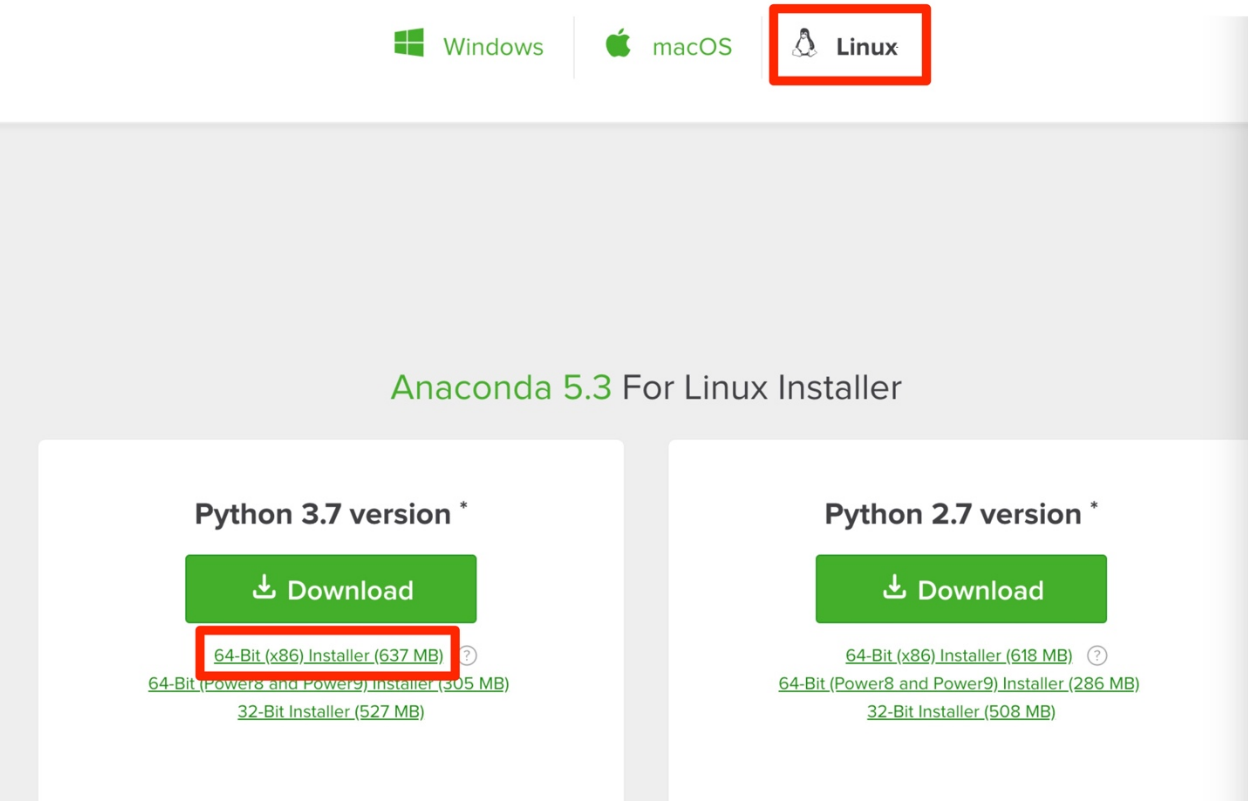
codeblocks

Code::Blocks IDE ★★★★★

Code::Blocks is a cross-platform Integrated Development Environment (IDE). It is based on a self-developed plugin framework allowing unlimited extensibility. Most of its fun...

Installed

Download anaconda with python 3.7 Version 64-Bit(x86) installer



Anaconda 5.3 For Linux Installer

Python 3.7 version *

Download

64-Bit (x86) Installer (637 MB)
64-Bit (Power8 and Power9) Installer (305 MB)
32-Bit Installer (527 MB)

Python 2.7 version *

Download

64-Bit (x86) Installer (618 MB)
64-Bit (Power8 and Power9) Installer (286 MB)
32-Bit Installer (508 MB)

run anaconda script

(https://www.digitalocean.com/community/tutorials/how-to-install-anaconda-on-ubuntu-18-04-quickstart)

```
bash Anaconda3-5.3.0-Linux-x86_64.sh
source ~/.bashrc
```

start anaconda

anaconda-navigator

7. Appendix

videos:

1.How to set up hpc clusters on CentOS?

https://www.youtube.com/watch?v=WgUjghaI_Ls&index=1&list=PLPx62H67wgD47MWNeAkvWjZURgpl6mBtu

https://www.youtube.com/watch?v=3MZcRBOsNWE&index=6&list=PLPx62H67wgD47MWNeAkvWjZURgpl6mBtu&t=1228s

2.How to Install Anaconda Python, Jupyter Notebook, Spyder on Ubuntu 18.04 Linux

https://www.youtube.com/watch?v=DY0DB_NwEu0

3. CUDA 9.0 installation in Ubuntu 16.4 + / Linux Mint - Full instruction with verification

https://www.youtube.com/watch?v=FK1y7XQuhp0&index=8&list=PLPx62H67wgD47MWNeAkvWjZURgpl6mBtu

websites:

1. How to configure openmpi with CUDA
   https://www.open-mpi.org/faq/?category=buildcuda

2. How to install anaconda on Ubuntu

   https://www.digitalocean.com/community/tutorials/how-to-install-anaconda-on-ubuntu-18-04-quickstart

3. OPENMPI FAQ
   https://www.open-mpi.org/faq/