CUDA_examples REPOSITORY TUTORIAL:

Author do recommend such logical sequence at development of CUDA program:
1) buy book „CUDA by examples” et. al. Sanders,
2) write few dozens of your own programs,
3) get an online Youtube course „Udacity: intro to parallel programming”,
4) write few dozens of your own programs,
5) buy book „Profesional CUDA programming” et. al. Cheng.

This step by step tutorial is intendent for executing repository CUDA programs. It consist of tip dashes:

1) obtain computer/laptop/tablet/phone with included NVidia GPU graphics (programs are written in Nvidia corporation C extensions language with some C++11 capabilities - Compute Unified Device Architecture CUDA). If you have not any, I will provide you some new-parts proposition in 2017y:

- CPU: Intel Celeron G3900
- MB: Gigabyte GA-B150-HD3 (note PCIe 3.0 x16 bandwidth ~12.6GBps)
- RAM: 1x GOODRAM PLAY Blue 2 / 4GB,
- HDD: 120GB SSD GOODRAM, or two pendrives 16GB with Lubuntu desktop 64bit live session on first pendrive, power supply,
- GPU: Nvidia GT730 4GB GDDR5 / GTX 1030 2GB
  - tip: consider operating system distribution booting to RAM (f.e. modified Ubuntu),
  - tip: consider good quality monitors, chair and keyboard (for my best knowledge mouse is unnecessary and slows work),
  - tip: for parallel cluster consider adding SSD RAID 0 (#mdadm are not so computationally expensive, as you think, especially for multicore CPU) and sufficient network connection,
  - tip: theoretical lower price – any used Personal Computer with PCIe 2.0x16 (f.e. Core2Quad + 2x2GB RAM + **GTX670** 2GB / **GTX780** 3GB). I did not check that configuration by myself, but should work (I have noticed non-compability of GPU with Dell R910, probably iDRAC issues),
  - tip: if you disable graphical server (#sudo service lightdm stop), you will get additional free space of +0.5GB RAM +0.34GB GPU memory, (author do recommend vim + screen in two windows), additionally some programs might execute faster,
  - tip: making software RAID 0 on hardware SSD RAID 0 disk array controller for big-data efficient streaming processing.

2) further consideration are for LINUX UBUNTU (checked for 16.04) - as long as it provides more efficient, and easier to understand system tools. *I have checked that on Microsoft Windows 10: Visual Studio Community (2015, great autoindent) and alternatively on Notepad++ with command line (shell) and both were working great (with different run commands).*

3) after signing in to operating system and making essential configurations (f.e. second sudo user for recovery mode) download CUDA toolkit from:

https://developer.nvidia.com/cuda-toolkit

4)to avoid:

- problems with lightdm log in (login loop)
- problems with driver istall ("Driver Installation failed: it appears, that a X server is running...")

via RUNFILE

5) and succesfully install a NVidia CUDA Toolkit on Ubuntu 16.04 64bit I have just had to do:

- login on live session on pendrive ("Try ubuntu, before install")

- add sudo user at live session:
    #sudo adduser admin (#pass: admin1)
    #sudo usermod -aG sudo admin
- logout from live session, log in as #admin
- download CUDA Toolkit from NVidia official site (~1.5GB)
- change privileges for downloaded installer file (DO NOT INSTALL AT THIS STEP!):
    #sudo chmod +x cuda_X.X.run
- switch to console view:
    #Ctr+Alt+F1 (to switch on terminal view)
    #Ctr+Alt+F7 (to switch from terminal view to graphical server)
- at console view (Ctr+Alt+F1) log in:
    #login: admin
    #pass: admin1
- stop graphical running service:
    #sudo service lightdm stop
- check if graphical server is off - after switching Ctr+Alt+F7 the monitor should be blank black, switch back on console view Ctr+Alt+F1
- install CUDA Toolkit, with such configuration:
    #sudo ./cuda_X.X.run
    #(press 'q' for license read skip)
    #do not install OpenGL library (I do not know why – please do not ask)
    #do not update system X configuration
    #other options make yes and paths as default
- turn on graphical server:
    #sudo service lightdm start
- log in as user (if you automatically log in as #ubuntu at live session log out):
    #login: admin
    #pass: admin1
- check if nvcc exists:
    # sudo find /usr/ -name 'nvcc'

6) obtain git:
    #sudo apt-get install git
7) clone my repo in home directory (~):
    #git clone https://github.com/PiotrLenarczykAnonim/CUDA_examples.git
8) check whatever nvcc compiler works:
    #cd CUDA_examples/01_makeSimple/
9) most of folders are configured for #make via BASH scripts:
    #./RUN_COMMANDS.sh
10) Thrust library is delivered with CUDA Toolkit, but I strongly recommend for cloning repo Mr Jared Hoberock with examples:
    # git clone https://github.com/thrust/thrust.git
11) there are another repo with C++ several useful examples:
    #git clone https://github.com/PiotrLenarczykAnonim/C-_examples.git

via .DEB PACKAGE

5deb) download download cudaXX.deb package from
    https://developer.nvidia.com/cuda-toolkit
6deb) in console:
- switch to console view:
    #Ctr+Alt+F1 (to switch on terminal view)
    #Ctr+Alt+F7 (to switch from terminal view to graphical server)
- at console view (Ctr+Alt+F1) log in:

#login: admin
#pass: admin1

- stop graphical running service:
#sudo service lightdm stop
- check if graphical server is off - after switching Ctr+Alt+F7 the monitor should be blank black, switch back on console view Ctr+Alt+F1

#sudo chmod +x cudaXX.deb
#sudo dpkg -i cudaXX.deb
#sudo apt-get update
#sudo aptitude install cuda g++-4.9 gcc-4.9
#sudo ln -s /usr/bin/g++-4.9 /usr/local/cuda/bin/g++
#sudo ln -s /usr/bin/gcc-4.9 /usr/local/cuda/bin/gcc
#sudo service lightdm start

7deb) similar to above 6) – 11)
8) runs without stopping graphical server on Ubuntu Desktop 17.04 64bit

if one poses devices with 6.1 and 3.5 computing capabilities in single Personal Computer one should reinstall cuda package proper to lower device compability (in my case it is a 3.5, and cuda-8-0) (“https://askubuntu.com/questions/959835/how-to-remove-cuda-9-0-and-install-cuda-8-0-instead”):

#dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 sudo dpkg --purge
#dpkg --install cuda**.deb
#sudo apt-get update
#sudo apt-get install cuda-X-X

Post Scriptum: Mostly I have been developing it on Dell Inspiron 7746 with i7 5500U, 16GB RAM 1666MHz, Nvidia GM108M GeForce 845M PCIe 3.0x4 2GB and SSD 1TB (OS: LINUX Ubuntu Desktop 16.04 64bit; LINUX Ubuntu Desktop 16.10 64bit). Also it was checked on PC’s: Intel i7 2660k, 16GB 2 channel RAM 1333MHz, Nvidia GK110 GTX780 3GB PCIe 2.0x16, SSD 0.24TB (OS: LINUX Ubuntu 16.04 64bit) and Intel Celeron G3900, 12GB 2 channel RAM DDR3 1333MHz, Nvidia GK110 GTX780 3GB PCIe 3.0x16, Nvidia GP108 GT1030 2GB PCIe 3.0x4.
Post Post Scriptum: If one will use customized liveCD from LINUX_tips repo, he or she should take into consideration that minimal recommended RAM size should be 8GB (or buy harddrive / pendrive).
Please do consider registers amount for some hardware:
GPR – General Purpose Register
FPR – Floating Point Register

Architecture, *or processor*	GPR	FPR
4004	1	0
16bit x86	8	8
x86 / IA-32 (*80386*)	8	8
x86-64(AMD 64)	16	16
IA-64 (Itanium)	128	128
Xeon Phi	16	32
CUDA	1	128*
Cray-1 GPR(data + addresses); FP(scalar * vector)	8+8=16	8*8=64
IBM POWER / POWERPC	32	32
IBM Cell	128	
SPARC	32	32
AVR MCU	32	0
ARM A32 / ARM T32	14	1-32
ARM A64	31	32
Epiphany	64**	
MIPS	32	32
*SW26010***	264	16.896

*3584 CUDA cores at 2nd Maxwell generation GP102 processor (GTX1080ti) – there are 1.835.008 FPR's
**64 cores at E64G401 – there is 4096 register acting as GPR's or FPR's,
*** amount approximated

GPU + global memory type	**GTX 780** GDDR5	**GT 1030** GDDR5
max GFLOPs (float 32 bit)	4000	942
measured GFLOPs (double 64 bit)	198	42
measured GFLOPs (float 32 bit)	3747	1310
measured Giops (integer 32bit)	790	440
Size of global memory [GB]	3	2
Bandwidth [GBps]	288.4	48
Practical throughput [GBps]	95	19
GFLOPs / PLN	5.33	3.14
Price PLN (1PLN = 0.28USD)	750	300
2channel DDR3 1333MHz has practical throughput of ~6.8GBps and theoretical bandwidth of 20GBps		

for your RAM details please refer to:
http://manpages.ubuntu.com/manpages/xenial/man1/mbw.1.html

MGPGPU (many GPU for the same price	-	3*GT730	-	-	4*GTX1080ti	-
GPU + global memory type	**GT 730** GDDR5	**GTX 780** GDDR5	**GT 1030** GDDR5	**GTX 1080ti GDDR5X**	**P6000** GDDR5X	***Intel Xeon E-2660*** *(16 HT cores; 2.2GHz) DDR3 (4-channel)*
Max. GFLOPs	690	4000	942	**10600**	10800	*166*
Size of global memory [GB]	4	3	2	**11**	24	*~4 (GB / core)*
Bandwidth [Tbps] (Throughput is about 50-60% of bandwidth)	0.04	0.2884	0.048	**0.484**	0.432	*0.05*
PCIe [2.0x16=x0.8; 3.0x16=x1; NVLink=x2]	0.8	1	1	**1**	2	*1*
Type [used=x1, brand-new=x1.2]	1.2	1	1.2	**1.2**	1.2	*1*
Slot size + Power (<45W)	1.2	1	1.2	**1**	1	*1*
GFLOPs / USD	0.8	1.5	0.9	**0.9**	0.3	*0.1*
Max power [W]	30	250	30	**250**	250	*95*
GFLOPs / W	23.00	16.00	31.4	**42.40**	43.20	*1.75*
Price USD	70	210	84	**924**	3360	*109*
normalized to P6000: GFLOPs*throughputGBps*memSizeGB / USD	23.74	62.02	6.75	**191.53**	62.71	*0.28*

Please do refer to:
https://asteroidsathome.net/boinc/cpu_list.php
http://cuda-z.sourceforge.net/
https://github.com/PiotrLenarczyk/CPlusPlus_examples/tree/master/78_linpackGFLOPS