Chapter 1: Introducing CUDA and Getting Started with CUDA

```
---
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 1 copy engine(s)
Run time limit on kernels: Yes
Integrated CPU 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
Supports Cooperative Kernel Launch: No
Supports MultiDevice Co-op Kernel Launch: No
Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
Compute Mode:
  < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
---
deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.0, CUDA Runtime Version = 9.0, NumDevs = 1
Result = PASS
bhaumik@bhaumik-Lenovo-ideapad-520-15IKB:~/NVIDIA_CUDA-9.0_Samples/1_Utils/Utils/deviceQuery$
```

```
```
Chapter 2: Parallel Programming using CUDA C

```
C:\WINDOWS\system32\cmd.exe
1 + 4 = 5
Press any key to continue . . .
```

```
C:\WINDOWS\system32\cmd.exe
Passing Parameter by Reference Output: 1 + 4 = 5
Press any key to continue . . .
```

```
C:\WINDOWS\system32\cmd.exe
Hello!!!!I'm thread in block: 2
Hello!!!!I'm thread in block: 7
Hello!!!!I'm thread in block: 6
Hello!!!!I'm thread in block: 8
Hello!!!!I'm thread in block: 1
Hello!!!!I'm thread in block: 0
Hello!!!!I'm thread in block: 5
Hello!!!!I'm thread in block: 10
Hello!!!!I'm thread in block: 9
Hello!!!!I'm thread in block: 11
Hello!!!!I'm thread in block: 4
Hello!!!!I'm thread in block: 3
Hello!!!!I'm thread in block: 14
Hello!!!!I'm thread in block: 13
Hello!!!!I'm thread in block: 12
Hello!!!!I'm thread in block: 15
All threads are finished!
Press any key to continue . . .
```
Detected 1 CUDA Capable device(s)

Device 0: "GeForce 940MX"
CUDA Driver Version / Runtime Version 9.1 / 9.0
CUDA Capability Major/Minor version number: 5.0
Total amount of global memory: 4896 MBytes (4294967296 bytes)
(3) Multiprocessors GPU Max Clock rate: 1189 MHz (1.19 GHz)
Memory Clock rate: 2505 Mhz
Memory Bus Width: 64-bit
L2 Cache Size: 1848576 bytes
Maximum Texture Dimension Size (x,y,z): 1D-(65536), 2D-(65536, 65536), 3D-(4096, 4096, 4096)
Maximum Layered 1D Texture Size, (num) layers 1D-(16384), 2048 layers
Maximum Layered 2D Texture Size, (num) layers 2D-(16384, 16384), 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 1 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
CUDA Device Driver Mode (TCC or WDDM): WDDM (Windows Display Driver Model)
Device supports Unified Addressing (UVA): Yes
Supports Cooperative Kernel Launch: No
Supports MultiDevice Co-op Kernel Launch: No
Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0

< Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
Press any key to continue . . .

Vector addition on CPU
The sum of 0 element is 0 + 0 = 0
The sum of 1 element is 2 + 1 = 3
The sum of 2 element is 8 + 2 = 10
The sum of 3 element is 18 + 3 = 21
The sum of 4 element is 32 + 4 = 36
Press any key to continue . . .
Vector addition on GPU
The sum of 0 element is 0 + 0 = 0
The sum of 1 element is 2 + 1 = 3
The sum of 2 element is 8 + 2 = 10
The sum of 3 element is 18 + 3 = 21
The sum of 4 element is 32 + 4 = 36
Press any key to continue . . .

No of Elements in Array:10000000
Device time 0.001000 seconds
host time 0.025000 Seconds
Press any key to continue . . .

Square of Number on GPU
The square of 0.000000 is 0.000000
The square of 1.000000 is 1.000000
The square of 2.000000 is 4.000000
The square of 3.000000 is 9.000000
The square of 4.000000 is 16.000000
Press any key to continue . . .
Chapter 3: Threads, Synchronization, and Memory
Array in Global Memory is:
At Index: 0 --> 0
At Index: 1 --> 1
At Index: 2 --> 2
At Index: 3 --> 3
At Index: 4 --> 4
Press any key to continue . . .

Use of Local Memory on GPU:
Value of Local variable in current thread is: 0
Value of Local variable in current thread is: 5
Value of Local variable in current thread is: 10
Value of Local variable in current thread is: 15
Value of Local variable in current thread is: 20
Press any key to continue . . .

Use of Shared Memory on GPU:
The running average after 0 element is 0.000000
The running average after 1 element is 0.500000
The running average after 2 element is 1.000000
The running average after 3 element is 1.500000
The running average after 4 element is 2.000000
The running average after 5 element is 2.500000
The running average after 6 element is 3.000000
The running average after 7 element is 3.500000
The running average after 8 element is 4.000000
The running average after 9 element is 4.500000
Press any key to continue . . .
10000 total threads in 100 blocks writing into 10 array elements
Number of times a particular Array index has been incremented without atomic add is:
index: 0 --> 1000 times
index: 1 --> 1000 times
index: 2 --> 1000 times
index: 3 --> 1000 times
index: 4 --> 1000 times
index: 5 --> 1000 times
index: 6 --> 1000 times
index: 7 --> 1000 times
index: 8 --> 1000 times
index: 9 --> 1000 times
Press any key to continue . . .
Use of Constant memory on GPU
The expression for input 0.000000 is 20.000000
The expression for input 1.000000 is 22.000000
The expression for input 2.000000 is 24.000000
The expression for input 3.000000 is 26.000000
The expression for input 4.000000 is 28.000000
Press any key to continue . . .

Use of Texture memory on GPU:
Texture element at 0 is : 0.000000
Texture element at 1 is : 1.000000
Texture element at 2 is : 2.000000
Texture element at 3 is : 3.000000
Texture element at 4 is : 4.000000
Texture element at 5 is : 5.000000
Texture element at 6 is : 6.000000
Texture element at 7 is : 7.000000
Texture element at 8 is : 8.000000
Texture element at 9 is : 9.000000
Press any key to continue . . .

The computed dot product is: 1047552.000000
The dot product computed by GPU is correct
Press any key to continue . . .

\[
\begin{bmatrix}
0 & 1 & 2 & 3 \\
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
3 & 3 & 3 & 3
\end{bmatrix} \times \begin{bmatrix}
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 3 \\
0 & 1 & 2 & 3
\end{bmatrix} = \begin{bmatrix}
0*0+0*0+0*0+0*0+0*0+0*0+0*0+0*0 & 0 & 0 & 0 \\
1*0+1*0+1*0+1*0+1*0 & 4 & 8 & 12 \\
2*0+2*0+2*0+2*0 & 8 & 16 & 24 \\
3*0+3*0+3*0+3*0 & 12 & 24 & 36
\end{bmatrix}
\]

\[
\begin{bmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{bmatrix} \rightarrow \begin{bmatrix} M_{00} & M_{01} & M_{10} & M_{11} \end{bmatrix}
\]
The result of Matrix multiplication is:

0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
0.000000  6.000000  12.000000  18.000000  24.000000  30.000000
0.000000  12.000000  24.000000  36.000000  48.000000  60.000000
0.000000  18.000000  36.000000  54.000000  72.000000  90.000000
0.000000  24.000000  48.000000  72.000000  96.000000  120.000000
0.000000  30.000000  60.000000  90.000000  120.000000  150.000000

Press any key to continue . . .
Chapter 4: Advanced Concepts in CUDA

[Image]

Time to add 50000 numbers: 0.9 ms
Vector addition on GPU
GPU has computed Sum Correctly
Press any key to continue...
Copy Engine

Stream 0: Memory Copy h a0
Stream 1: Memory Copy h a1
Stream 0: Memory Copy h b0
Stream 1: Memory Copy h b1
Stream 0: Memory Copy h c0
Stream 1: Memory Copy h c1

Kernel Engine

Stream 0: gpuAdd Kernel
Stream 1: gpuAdd Kernel

C:\WINDOWS\system32\cmd.exe

Time to add 100000 numbers: 0.9 ms
Vector addition on GPU
GPU has computed Sum Correctly
Press any key to continue...

C:\WINDOWS\system32\cmd.exe

The Enumeration sorted Array is:
3
4
5
8
9
Press any key to continue...
Histogram using 16 bin without shared Memory is:

bin 0: count 63
bin 1: count 63
bin 2: count 63
bin 3: count 63
bin 4: count 63
bin 5: count 63
bin 6: count 63
bin 7: count 63
bin 8: count 62
bin 9: count 62
bin 10: count 62
bin 11: count 62
bin 12: count 62
bin 13: count 62
bin 14: count 62
bin 15: count 62

Press any key to continue . . .

Press any key to continue . . .
Chapter 5: Getting Started with OpenCV with CUDA Support
OpenCV!
Performance of Thresholding on CPU:
Time: 0.169766
FPS: 5.89046

Performance of Thresholding on GPU:
Time: 0.000550593
FPS: 1816.22
Chapter 6: Basic Computer Vision Operations Using OpenCV and CUDA

```
bhaumik@bhaumik-Lenovo-ideapad-520-15IKB:~/Desktop/opencv/Chapter 6$ g++ -std=c++11 individual_pixel.cpp `pkg-config --libs --cflags opencv` -o pixel
bhaumik@bhaumik-Lenovo-ideapad-520-15IKB:~/Desktop/opencv/Chapter 6$ ./pixel
Pixel intensity of grayscale image at (100,50) is: 9
Pixel intensity of color image at (100,50) is: [175, 179, 177]
```
<table>
<thead>
<tr>
<th>Averaging Filter 3x3</th>
<th>Averaging Filter 5x5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{1}{9} ) \ast \begin{bmatrix} 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 \end{bmatrix}</td>
<td>( \frac{1}{25} ) \ast \begin{bmatrix} 1 &amp; 1 &amp; 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 &amp; 1 &amp; 1 \ 1 &amp; 1 &amp; 1 &amp; 1 &amp; 1 \end{bmatrix}</td>
</tr>
</tbody>
</table>
Gaussian Filter 5x5

\[
\frac{1}{273} * \begin{bmatrix}
1 & 4 & 7 & 4 & 1 \\
4 & 16 & 26 & 16 & 4 \\
7 & 26 & 41 & 26 & 7 \\
4 & 16 & 26 & 16 & 4 \\
1 & 4 & 7 & 4 & 1 \\
\end{bmatrix}
\]
\[ S_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]
$S_x = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} \quad S_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$
Chapter 7: Object Detection and Tracking Using OpenCV and CUDA
\[ |H_a| = D_x D_y - (wDxy)^2 \]
FPS : 112.059
FPS : 113.165
FPS : 111.566
FPS : 113.295
FPS : 113.228
Chapter 8: Introduction to the Jetson TX1 Development Board and Installing OpenCV on Jetson TX1
JetPack

NVIDIA JetPack SDK is the most comprehensive solution for building AI applications. Use the JetPack installer to flash your Jetson Developer Kit with the latest OS image, to install developer tools for both host PC and Developer Kit, and to install the libraries and APIs, samples, and documentation needed to jumpstart your development environment.

JetPack 3.3 with the latest BSPs (L4T 28.2.1 for Jetson TX2/TX2i and L4T 28.2 for Jetson TX1) is the latest production software release for NVIDIA Jetson TX2, Jetson TX2i, and Jetson TX1. It bundles all the Jetson platform software, including TensorFlow, cuDNN, CUDA Toolkit, VisionWorks, GStreamer, and OpenCV, all built on top of L4T with LTS Linux kernel.

The highlight of this release is TensorRT 4.0, enabling support for TensorFlow’s TensorRT integration feature. Additionally, cuDNN has a small point release to support the new TensorRT version, while all other JetPack components remain unchanged from JetPack 3.2.1.

View the full 3.3 Release Notes here.

[ Download Jetpack ]
bhaumik@bhaumik-VirtualBox:~$ cd Desktop/jetpack
bhaumik@bhaumik-VirtualBox:~Desktop/jetpack$ chmod +x JetPack-L4T-3.3-linux-x64_b39.run
bhaumik@bhaumik-VirtualBox:~Desktop/jetpack$ ./JetPack-L4T-3.3-linux-x64_b39.run

Creating directory _installer
Verifying archive integrity... All good.
Uncompressing JetPack 57%

JetPack 3.3 Installer

NVIDIA JetPack SDK is the most comprehensive solution for building AI applications. It bundles all the Jetson platform software, including TensorRT, cuDNN, CUDA Toolkit, VisionWorks, GStreamer, and OpenCV, all built on top of L4T with LTS Linux kernel.

Use the JetPack Installer to flash your Jetson Developer Kit with the latest OS image, to install developer tools for both host PC and Developer Kit, and to install the libraries and APIs, samples, and documentation needed to jumpstart your development environment.
Installation Configuration

Please specify the directory where JetPack 3.3 will be installed.
Installation Directory: /home/bhaumik/Desktop/jetpack/

Please specify the directory where the components will be downloaded.
Download Directory: /home/bhaumik/Desktop/jetpack/jetpack_download

Select Development Environment
Please specify the development environment you would like to set up:

- Jetson TX2
- Jetson TX2i
- Jetson TX1
## JetPack 3.3 Components Manager

<table>
<thead>
<tr>
<th>Package</th>
<th>Installed Version</th>
<th>Size</th>
<th>Action</th>
<th>Progress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host - Ubuntu</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tegra Graphics Debugger</td>
<td></td>
<td>172MB</td>
<td>install</td>
<td>5m 50s remaining - 6.5% (754 KB/s)</td>
</tr>
<tr>
<td>NVIDIA System Profiler</td>
<td></td>
<td>185MB</td>
<td>install 4.0</td>
<td>3m 7s remaining - 7.2% (441 KB/s)</td>
</tr>
<tr>
<td>JetPack Documentation</td>
<td></td>
<td>20MB</td>
<td>install 3.3</td>
<td>1m 20s remaining - 66.7% (7 KB/s)</td>
</tr>
<tr>
<td>DevTools Documentation</td>
<td></td>
<td>977KB</td>
<td>install 3.3</td>
<td>.....</td>
</tr>
<tr>
<td>CUDA Toolkit</td>
<td></td>
<td>3254MB</td>
<td>install 9.0</td>
<td>.....</td>
</tr>
<tr>
<td>Target - Jetson TX1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux for Tegra Host Side Imaging</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>File System and OS</td>
<td></td>
<td>2463MB</td>
<td>install 28.2</td>
<td>.....</td>
</tr>
<tr>
<td>Drivers</td>
<td></td>
<td>131MB</td>
<td>install 28.2</td>
<td>.....</td>
</tr>
<tr>
<td>Flash OS Image to Target</td>
<td></td>
<td>4820MB</td>
<td>install 28.2</td>
<td>.....</td>
</tr>
<tr>
<td>Install on Target</td>
<td></td>
<td>578MB</td>
<td>install 9.0</td>
<td>.....</td>
</tr>
<tr>
<td>CUDA Toolkit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compile CUDA Samples</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cuDNN Package</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TensorFlow</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multimedia API package</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenCV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Description
- Disk Space
- Terminal

- Automatically resolve dependency conflicts

### Installing

Host installation is complete. Installer will continue with target hardware setup.

Completed

Click next to proceed
Network Layout

Please select the network layout.

- Device accesses Internet via router/switch

- Device accesses Internet via host machine through setting up a new DHCP server configuration on host.

Network Interface Selection

Please select the network interface on host that connects to the same router/switch as the target device.

eth0  

JetPack requires this information to setup ssh connection between target and host.
Post Installation

Please put your device to Force USB Recovery Mode, when you are ready, press Enter key.

To place system in Force USB Recovery Mode:
1. Power down the device. If connected, remove the AC adapter from the device. The device MUST be powered OFF, not in a suspend or sleep state.
2. Connect the Micro-B plug on the USB cable to the Recovery (USB Micro-B) Port on the device and the other end to an available USB port on the host PC.
3. Connect the power adapter to the device.
4. Press and release the POWER button to power on device. Press and hold the FORCE RECOVERY button: while pressing the FORCE RECOVERY button, press and release the RESET button; wait two seconds and release the FORCE RECOVERY button.
5. When device is in recovery mode, lsusb command on host will list a line of "Nvidia Corp"

Post Installation

populating extlinux.conf.emmc to rootfs... done.
populating /home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/kernel/db/tb/tegra210-jetson-tx1-p2597-2180-a01-devkit.dbt to rootfs... done.
done.
Making Boot image... done.
copying bcffile(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/bootloader/t210ref/cfg/board_config_p2597-devkit.xml)... done.
Existing sosfile(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/bootloader/nvboot_recovery.bin) reused.
copying tegraboot(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/bootloader/t210ref/nvboot.bin)... done.
Existing bpfile(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/bootloader/bpmp.bin) reused.
copying wdboot(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/bootloader/t210ref/warmboot.bin)... done.
Existing tosfile(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/bootloader/tos.img) reused.
Existing ekfile(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/bootloader/eks.img) reused.
copying dbfile(/home/bhaumik/Desktop/jetpack/64_TX1/Linux_for_Tegra/kernel/db/tb/tegra210-jetson-tx1-p2597-2180-a01-devkit.dbt)... done.
Making system.img...

Post Installation Jetson TX1

Following actions will be performed at this stage.
- Push and install 64Bit CUDA on target
- Push and install TensorRT on target
- Push and install 64Bit OpenCV on target
- Push and install 64Bit cuDNN on target
- Push and install 64Bit VisionWorks on target
- Push and install 64Bit VisionWorks SFM on target
- Push and install 64Bit VisionWorks Tracking on target
- Cross-compile 64Bit CUDA samples and push to target
- Push and Install MMAP on target
Installing CUDA on target

Copying /home/bhaumik/Desktop/jetpack/jetpack_download/cuda-repo-l4t-9-0-local_9.0.252-1_arm64.deb file to target...
cuda-repo-l4t-9-0-local_9.0.252-1_arm64.deb
343,179,264 56% 7.07MB/s 0:00:36

JetPack 3.3

Installation Complete

JetPack 3.3 installation has completed successfully.

☐ Remove downloaded files
Chapter 9: Deploying Computer Vision Applications on Jetson TX1

```
$ nvcc kernel.cu -o device
$ ./device

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

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA Tegra X1"
CUDA Driver Version / Runtime Version : 9.0 / 9.0
CUDA Capability Major/Minor version number: 5.3
Total amount of global memory: 3984 MBytes (4177342464 bytes)
(2) Multiprocessors  GPU Max Clock rate: 998 MHz (1.00 GHz)
Memory Clock rate: 13 MHz
Memory Bus Width: 64-bit
L2 Cache Size: 262144 bytes
Maximum Texture Dimension Size (x,y,z)  1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 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: 32768
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 1 copy engine(s)
Run time limit on kernels: 
Integrated GPU sharing Host Memory: Yes
Support host page-locked memory mapping: Yes
Alignment requirement for Surfaces: Yes
Device has ECC support: DlEnabled
Device supports Unified Addressing (UVA): Yes
Supports Cooperative Kernel Launch: No
Supports MultiDevice Co-op Kernel Launch: No
Device PCI Domain ID / Bus ID / location ID: 0 / 0 / 0
Compute Mode: < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

```
$ nvcc 01_performance_cuda_events.cu -o gpu_add
$ ./gpu_add

Time to add 50000 numbers: 3.4 ms
Vector addition on GPU
GPU has computed Sun Correctly
```
Performance of Addition on Jetson TX!
Time: 0.000262104
FPS: 3815.28

Performance of Thresholding
Time: 0.000326671
FPS: 3061.18
FPS : 45.9451
FPS : 38.3942
FPS : 39.1694
FPS : 39.365
FPS : 38.7298
FPS : 34.2839
FPS : 75.2029
FPS : 59.0898
FPS : 47.6766
FPS : 49.3542
FPS : 34.7506
FPS : 38.3625
FPS : 38.4274
FPS : 44.3814
FPS : 77.2785
FPS : 65.9443
FPS : 58.8832
FPS : 54.1984
FPS : 49.1819
FPS : 78.8699
FPS : 71.9925
FPS : 65.1592
FPS : 60.9173
Chapter 10: Getting Started with PyCUDA

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
- Version: 10, 8.1, 7, Server 2016, Server 2012 R2
- Installer Type: exe (network), exe (local)

Download Installer for Windows 10 x86_64

The base installer is available for download below.

Base Installer

Download (1.5 GB)

Installation Instructions:
1. Double click cuda_9.2.148_win10.exe
2. Follow on-screen prompts
PS C:\Users\bbaum\Downloads> `pip install -p /ycuda-2017.1.1-cuda92148-cp36-cp36m-win_amd64.whl` Processing c:\Users\bbaum\Downloads\pycuda-2017.1.1-cuda92148-cp36-cp36m-win_amd64.whl Requirement already satisfied: pytools>=2.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: decorator>=3.2.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: appdirs>=1.4.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: py>=1.5.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: six>=1.10.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: attrs>=17.2.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: pluggy<0.7,>=0.3 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: colorama in c:\programdata\anaconda3\lib\site-packages (from pycuda==2017.1.1+cuda92148) Requirement already satisfied: numpy>=1.6.0 in c:\programdata\anaconda3\lib\site-packages (from pycuda==2011.2->pycuda==2017.1.1+cuda92148) Installing collected packages: pycuda Found existing installation: pycuda-2017.1.1+cuda9185 Uninstalling pycuda-2017.1.1+cuda9185 Successfully uninstalled pycuda-2017.1.1+cuda9185
You are using pip version 9.0.1 however version 18.0 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

In [4]: import pycuda

In [5]:
## Select Target Platform

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

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Windows</th>
<th>Linux</th>
<th>Mac OSX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>x86_64</td>
<td>ppc64le</td>
<td></td>
</tr>
<tr>
<td>Distribution</td>
<td>Fedora</td>
<td>OpenSUSE</td>
<td>RHEL</td>
</tr>
<tr>
<td>Version</td>
<td>17.10</td>
<td>16.04</td>
<td></td>
</tr>
<tr>
<td>Installer Type</td>
<td>runfile (local)</td>
<td>deb (local)</td>
<td>deb (network)</td>
</tr>
</tbody>
</table>

## Download Installer for Linux Ubuntu 17.10 x86_64

The base installer is available for download below.

![Base Installer](#)

**Base Installer**

**Download [1.7 GB](#)**

**Installation Instructions:**

1. Run `sudo sh cuda_9.2.148_396.37_linux.run`
2. Follow the command-line prompts
conda install -c lukepfister pycuda

Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.4.10
  latest version: 4.5.8

Please update conda by running

$ conda update -n base conda

## Package Plan ##

environment location: /home/bhaumik/anaconda3

added / updated specs:
- pycuda

The following packages will be downloaded:

<table>
<thead>
<tr>
<th>package</th>
<th>build</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca-certificates-2018.03.07</td>
<td>0</td>
<td>124 KB</td>
</tr>
<tr>
<td>pycuda-2017.1</td>
<td>py36_0</td>
<td>627 KB</td>
</tr>
<tr>
<td>appdirs-1.4.3</td>
<td>py36h28b3542_0</td>
<td>16 KB</td>
</tr>
<tr>
<td>pytools-2018.4</td>
<td>py36_0</td>
<td>110 KB</td>
</tr>
<tr>
<td>mako-1.0.4</td>
<td>py36_0</td>
<td>116 KB</td>
</tr>
<tr>
<td>openssl-1.0.20</td>
<td>h20670df_0</td>
<td>3.4 MB</td>
</tr>
<tr>
<td>certifi-2018.4.16</td>
<td>py36_0</td>
<td>142 KB</td>
</tr>
</tbody>
</table>

Total: 4.6 MB

In [4]: import pycuda
In [5]:
Chapter 11: Working with PyCUDA

```
PS C:\CUDA book code\Chapter11> python .\hello_pycuda.py
Hello, PyCUDA!!!
PS C:\CUDA book code\Chapter11>
```

```
Windows PowerShell

PS C:\CUDA book code\Chapter11> python .\thread_execution.py
```

```
In [11]: runfile('G:\Cuda opencv book material\CUDA book code\Chapter11\add_n.py', wdir='G:\Cuda opencv book material\CUDA book code\Chapter11')
Addition on GPU:
1.2273334 + 1.3404454 = 2.5677788
```

[67]
### Addition on GPU:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7673203 + 0.5080069</td>
<td>1.2753272</td>
</tr>
<tr>
<td>0.28488383 + 0.15324554</td>
<td>0.43812937</td>
</tr>
<tr>
<td>-0.3220178 + -0.10700232</td>
<td>-0.4290201</td>
</tr>
<tr>
<td>-0.501334 + -1.7047318</td>
<td>-2.206066</td>
</tr>
<tr>
<td>0.023053076 + 0.34796545</td>
<td>0.37101853</td>
</tr>
<tr>
<td>-0.44806996 + -2.071736</td>
<td>-2.5198061</td>
</tr>
<tr>
<td>2.9193559 + 0.7721601</td>
<td>3.691516</td>
</tr>
<tr>
<td>-0.8016763 + -0.31726292</td>
<td>-1.1189392</td>
</tr>
<tr>
<td>0.607628 + -1.2539302</td>
<td>-0.6463022</td>
</tr>
<tr>
<td>-0.80436313 + 1.2789425</td>
<td>0.47457933</td>
</tr>
</tbody>
</table>

```python
In [9]: runfile('G:/cude opencv book material/CUDA book code/Chapter11/add_number.py',
           wdir='G:/cude opencv book material/CUDA book code/Chapter11')
```

Addition of 1000000 element of GPU
0.009422s
Addition of 1000000 element of CPU
0.41515421867370605 s

### Time of Squaring on GPU without InOut

0.149003s

original array:

```
[[3. 2. 1. 4. 3.]
 [1. 1. 1. 2. 2.]
 [2. 1. 1. 3. 2.]
 [3. 4. 2. 3. 1.]
 [4. 1. 4. 3. 1.]]
```

Square with kernel:

```
[[9. 4. 1. 16. 9.]
 [1. 1. 1. 4. 4.]
 [4. 1. 1. 9. 4.]
 [9. 16. 4. 9. 1.]
[16. 1. 16. 9. 1.]]
```

Square with InOut:

```
[[9. 4. 1. 16. 9.]
 [1. 1. 1. 4. 4.]
 [4. 1. 1. 9. 4.]
 [9. 16. 4. 9. 1.]
[16. 1. 16. 9. 1.]]
```

Time of Squaring on GPU with InOut
0.004260s
original array:
[[1 1 3 1 4]
[4 3 4 2 1]
[4 1 1 4 4]
[2 3 2 4 1]
[2 1 2 4 3]]
Squared with gpuarray:
[[ 1.  1.  9.  1. 16.]
[16.  9. 16.  4.  1.]
[16.  1.  1. 16. 16.]
[ 4.  9.  4. 16.  1.]
[ 4.  1.  4. 16.  9.]]
Time of Squaring on GPU with gpuarray 0.058682s

In [3]: runfile('G:/cuda opencv book material/CUDA book code/Chapter11/gpu_dot.py',
, wdir='G:/cuda opencv book material/CUDA book code/Chapter11')
Answer of Dot Product using numpy
633.0
Time taken for Dot Product using numpy
0.03769350051879883 s
Answer of Dot Product on GPU
633.0
Time taken for Dot Product on GPU
0.000108s
The computed dot product is correct

In [3]: runfile('G:/cuda opencv book material/CUDA book code/Chapter11/gpu_dot.py',
, wdir='G:/cuda opencv book material/CUDA book code/Chapter11')
Answer of Dot Product using numpy
633.0
Time taken for Dot Product using numpy
0.03769350051879883 s
Answer of Dot Product on GPU
633.0
Time taken for Dot Product on GPU
0.000108s
The computed dot product is correct

\[
\begin{bmatrix}
4 & 1 & 1 \\
1 & 2 & 3 \\
1 & 1 & 3
\end{bmatrix}
+ \begin{bmatrix}
3 & 3 & 4 \\
2 & 3 & 3 \\
4 & 1 & 1
\end{bmatrix}
= \begin{bmatrix}
4 \cdot 3 + 1 \cdot 2 + 1 \cdot 4 & 16 & 20 \\
1 \cdot 3 + 2 \cdot 2 + 3 \cdot 4 & 12 & 13 \\
1 \cdot 3 + 1 \cdot 2 + 3 \cdot 4 & 9 & 10
\end{bmatrix}
\]

Matrix A:
[[4. 1. 1.]
 [1. 2. 3.]
 [1. 1. 3.]]

Matrix B:
[[3. 3. 4.]
 [2. 4. 4.]
 [4. 1. 1.]]

Matrix Multiplication result:
[[18. 18. 24.]
 [10. 12. 13.]
 [17. 20. 21.]]

The computed matrix multiplication is correct.


Addition of 1000000 element of GPU: 
0.000629s
The sum computed on GPU is correct.


Vector A
[0 1 2 3 4]
Vector B
[0 1 2 3 4]
The computed dot product using reduction:
30
Dot Product on GPU:
2.546338s

The input data:
[7 5 9 2 9 7 7 7 2 6]
The computed cumulative sum using Scan:
[ 7 12 21 23 32 39 46 53 55 61]
Cumulative Sum on GPU:
0.002032s
Chapter 12: Basic Computer Vision Applications Using PyCUDA

Time for Calculating Histogram on GPU without shared memory
0.001040s
Time for Calculating Histogram using OpenCV
0.001040s

Time for Calculating Histogram on GPU with shared memory
0.000839s