Ubuntu 16.04安装CUDA与cuDNN库

概述

CUDA是nvidia公司为自己GPU专门推出的并行程序设计语言，语法结构与C语言类似，CUDA能够让我们充分地利用GPU的计算能力，提高程序性能，但是并行程序设计与计算机硬件有极强的关联性。

环境准备

本文将在安装了nvidia tesla P40 GPU的高性能HP服务器中进行。下面给出了详细的系统配置。

项目	版本
操作系统	Ubuntu 16.04.4
gcc	gcc 5.4.0
CUDA	cuda 9.2
cuDNN	cuDNN for CUDA 9.2
内核版本	4.15.0-34-generic

注意：低版本的内核不支持安装CUDA，因此要尽量使用Ubuntu 16.04之后的操作系统。

安装步骤

安装显卡驱动

```
sudo apt-get update   #更新系统
sudo apt-get install build-essential
sudo apt-get update
sudo apt-get upgrade  #保持当前大版本，将小版本升级到最新
lspci |grep -i nvidia   #查看系统中安装的显卡版本，因为tesla是没有视频输出功能的GPU因此使用该命令查看

uname -r  #查看内核版本
sudo apt-get install linux-headers-${uname -r}   #安装内核头文件

ubuntu-drivers devices   #查看GPU版本以及对应的GPU驱动
sudo service lightdm stop #在安装驱动时，关闭图形界面
sudo ubuntu-drivers autoinstall  #安装上述命令中查看到的所有驱动，该命令很重要，必须要执行，执行完成之后才能使用下面的nvidia-smi指令
sudo reboot  #需要重启计算机才能正确识别显卡驱动
nvidia-smi   #查看显卡的信息，如下面两张图图，我们能够清楚的看到，系统中安装了两张显卡，分别是Quadro P400 （视频接口卡）和P40 （并行计算卡），在我们后续的实验中，将使用P40来做并行计算
nvidia-smi -L   #查看系统中安装的GPU名称
sudo service lightdm restart

#上述步骤使得nvidia-smi命令能够正常使用，接着需要在这里安装相应的GPU显卡的驱动https://www.nvidia.com/Download/index.aspx 在nvidia driver官网下载相应的显卡驱动，之后会将显卡驱动升级
sudo apt-key add /var/nvidia-diag-driver-local-repo-396.44/7fa2af80.pub
sudo dpkg -i nvidia-diag-driver-local-repo-ubuntu1604-396.44_1.0-1_amd64.deb  #telsa p40当前适合的显卡驱动

sudo apt-get update
sudo apt-get upgrade
sudo apt-get install mesa-utils   #安装查看是否安装了正确的GPU驱动
glxinfo | grep rendering   #如果返回信息为YES，则说明已经正确的安装了显卡驱动，但是如果为NO，则说明驱动没有正确的安装
```

```
nvidia-smi -q -i 1  #查看第一块GPU的详细信息，我们能够看到究竟第1块GPU的型号，这对于在运行Python程序时指定在某块具体的GPU上运行有极其重要的影响
nvidia-smi -q -i 0  #查看第一块GPU的详细信息，我们能够看到究竟第0块GPU的型号，这对于在运行Python程序时指定在某块具体的GPU上运行有极其重要的影响
```

更新GPU驱动之前，驱动的版本为：384.130

```
graph@graph-HP-Z8-G4-Workstation:~$ nvidia-smi
Tue Sep 11 14:16:06 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 384.130                Driver Version: 384.130                   |
|-------------------------------+----------------------+----------------------+
| 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 P400         Off  | 00000000:2D:00.0  On |                  N/A |
| 34%   45C    P8   ERR! /  N/A |    110MiB /  1990MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P40           Off  | 00000000:8D:00.0 Off |                  Off |
| N/A   35C    P8    10W / 250W |      0MiB / 24445MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1432      G   /usr/lib/xorg/Xorg                           108MiB |
+-----------------------------------------------------------------------------+
```

更新GPU驱动之后，驱动版本为：396.44

```
graph@graph-HP-Z8-G4-Workstation:~$ nvidia-smi
Tue Sep 11 14:32:36 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 396.44                 Driver Version: 396.44                    |
|-------------------------------+----------------------+----------------------+
| 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 P400         Off  | 00000000:2D:00.0  On |                  N/A |
| 35%   48C    P8    N/A /  N/A |    110MiB /  1991MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P40           Off  | 00000000:8D:00.0 Off |                  Off |
| N/A   41C    P8    11W / 250W |      0MiB / 24451MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    0      1430      G   /usr/lib/xorg/Xorg                           107MiB |
+-----------------------------------------------------------------------------+
```

安装CUDA

在官网中选择后执行下面的命令: https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&target_distro=Ubuntu&target_version=1604&t

arget_type=debnetwork

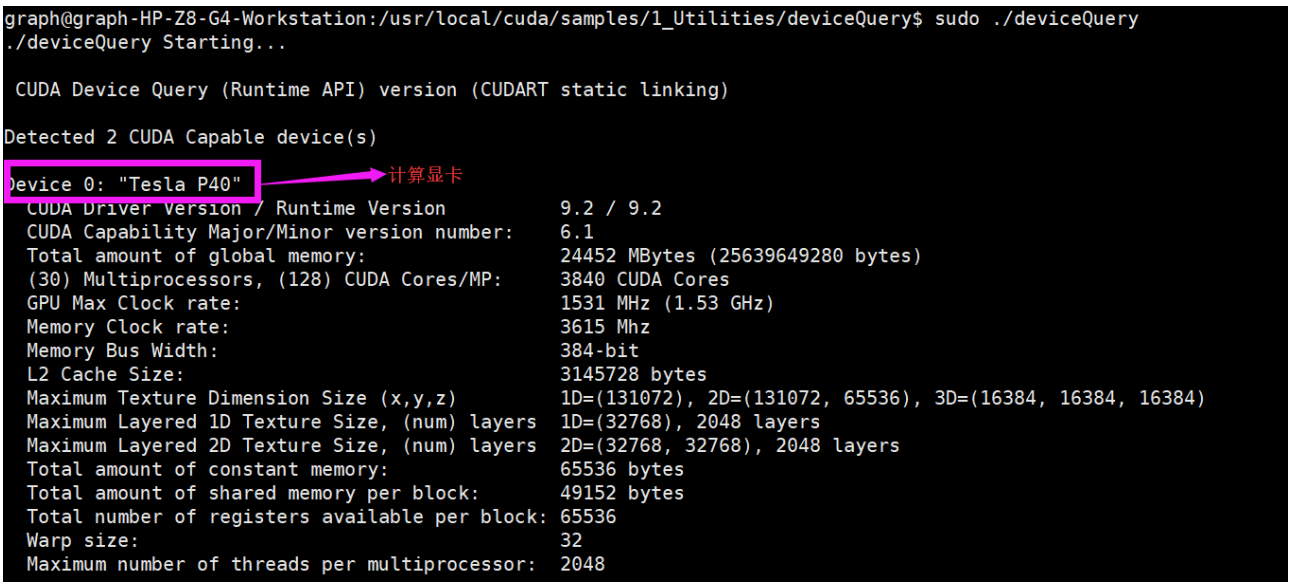
```
sudo dpkg -i cuda-repo-ubuntu1604_9.2.148-1_amd64.deb
sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa2af80.pub
sudo apt-get update
sudo apt-get install cuda
```

安装完成之后，我们能够看到在/usr/local下面有对应版本的cuda文件夹及其软连接，此时我们需要在环境变量~/.bashrc中添加 PATH=$PATH:/usr/local/cuda/bin 和 LD_LINRARY_PATH=/usr/local/cuda/lib64。

C程序测试CUDA的安装是否成功

在上一个阶段，我们已经成功的安装CUDA，但是还不知道安装是否成功，当然在安装的过程中，没有报错已经能够表明安装成功，但是仍然不够，下面，我们cuda/sample下自带的code示例测试是否成功安装。

```
cd /usr/local/cuda/samples/1_Utilities/deviceQuery
sudo make
sudo ./deviceQuery     #在输出中我们能够看到GPU的型号，像下面图示，说明成功安装CUDA支持
nvcc --version    #nvcc是CUDA程序的编译器，如果能够正确的返回nvcc的版本，也充分说明CUDA已经成功安装
```



安装cuDNN

我们能够在 https://developer.nvidia.com/cuda-gpus 上查找到关于Tesla p40的compute Capability为6.1，因此，为了加速神经网络的计算，应该安装cuDNN库来加速神经网络的计算过程。

```
#下载的安装包为：cudnn-9.2-linux-x64-v7.2.1.38.tgz
tar -xvf cudnn-9.2-linux-x64-v7.2.1.38.tgz  #解压后在下载的目录下能够看到一个名为cuda的文件夹
cd cuda
sudo cp lib64/lib* /usr/local/cuda/lib64
sudo cp include/cudnn.h /usr/local/cuda/include
cd /usr/local/cuda/lib64/
sudo chmod +r libcudnn.so.7.2.1
sudo ln -sf libcudnn.so.7.2.1 libcudnn.so.7
sudo ln -sf libcudnn.so.7 libcudnn.so
sudo ldconfig
```

至此完成cuDNN库的安装，它在计算能力3以上的计算机中，能够加速神经网络的计算。

nvidia-smi各个参数的含义

在安装好GPU和其驱动之后，采用nvidia-smi参数能够看到当前显卡的参数，其中Perf参数是我们最为关心的，如下图（下图中的数据采用命令 `watch -n 1 nvidia-smi` 来查看，该指令代表实时监控GPU的状态）：

```
graph@graph-HP-Z8-G4-Workstation:/devdata1/JunpengZhu/Papers/Technical-Documents/CUDA Programming/codes$ nvidia-smi
Tue Oct 16 17:28:02 2018
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.48                 Driver Version: 410.48                    |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla P40           Off  | 00000000:8D:00.0 Off |                  Off |
| N/A   47C    P0    48W / 250W |      0MiB / 24451MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   1  Tesla P4            Off  | 00000000:99:00.0 Off |                    0 |
| N/A   47C    P8     7W /  75W |      0MiB /  7611MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+
|   2  Quadro P400         Off  | 00000000:A6:00.0  On |                  N/A |
| 34%   41C    P8    N/A /  N/A |    258MiB /  1994MiB |      0%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|    2      1634      G   /usr/lib/xorg/Xorg                           155MiB |
|    2      2408      G   compiz                                       101MiB |
+-----------------------------------------------------------------------------+
```

图1 指定了使用Telsa P40来计算

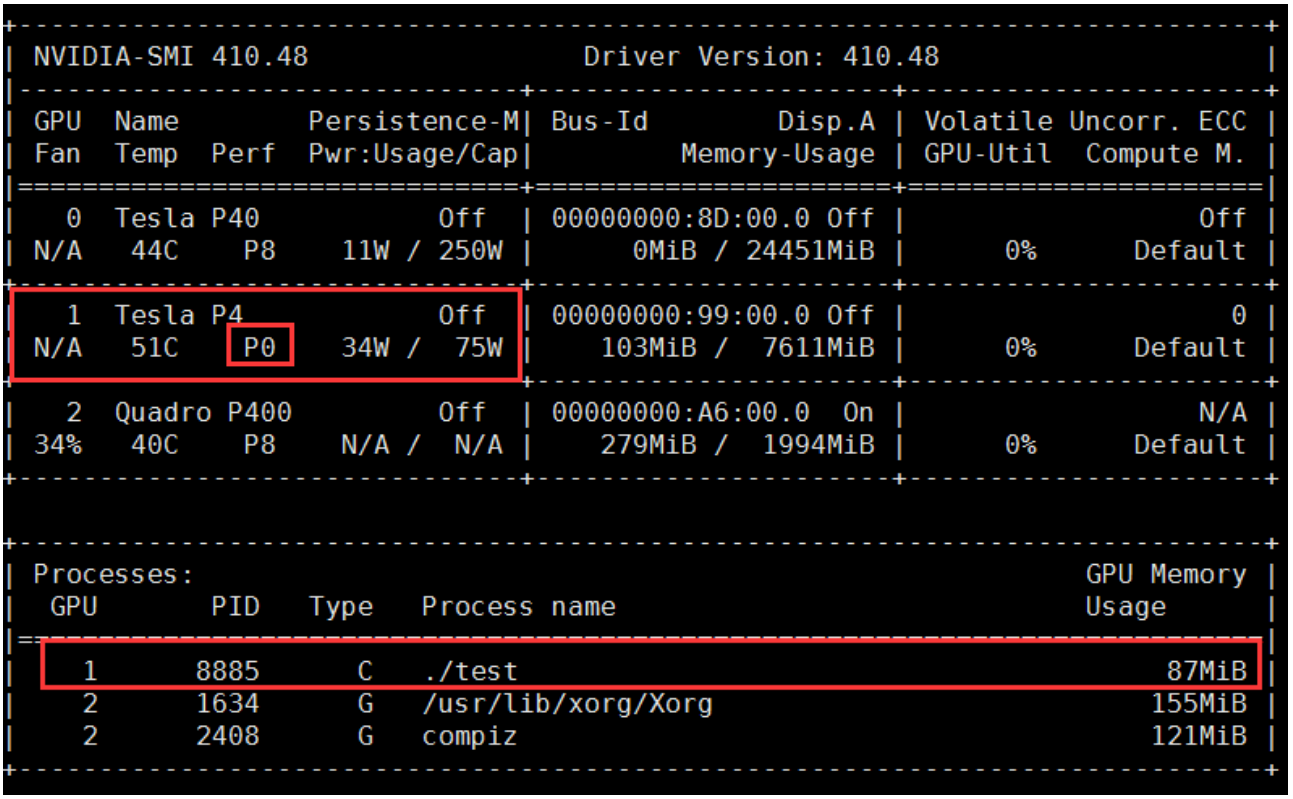


图2 指定了使用Telsa P4来计算

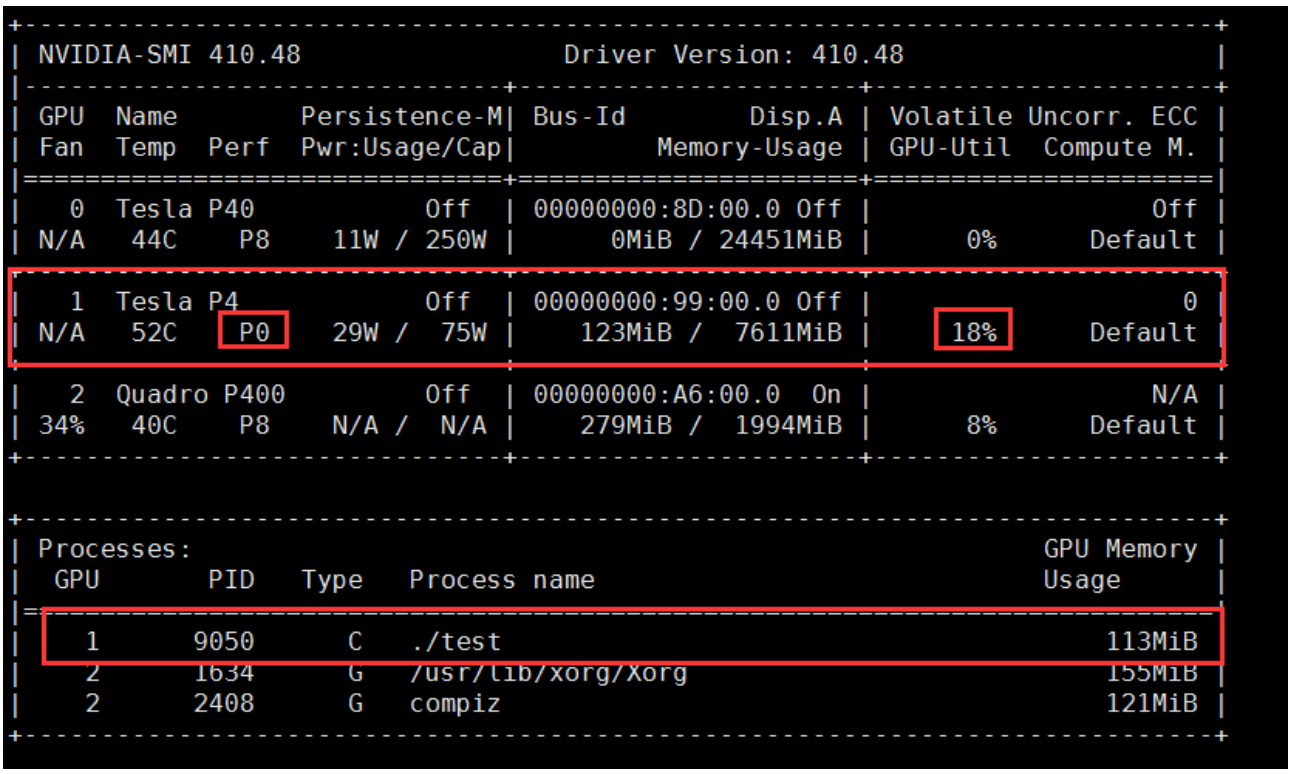


图3 指定了使用Telsa P4来计算

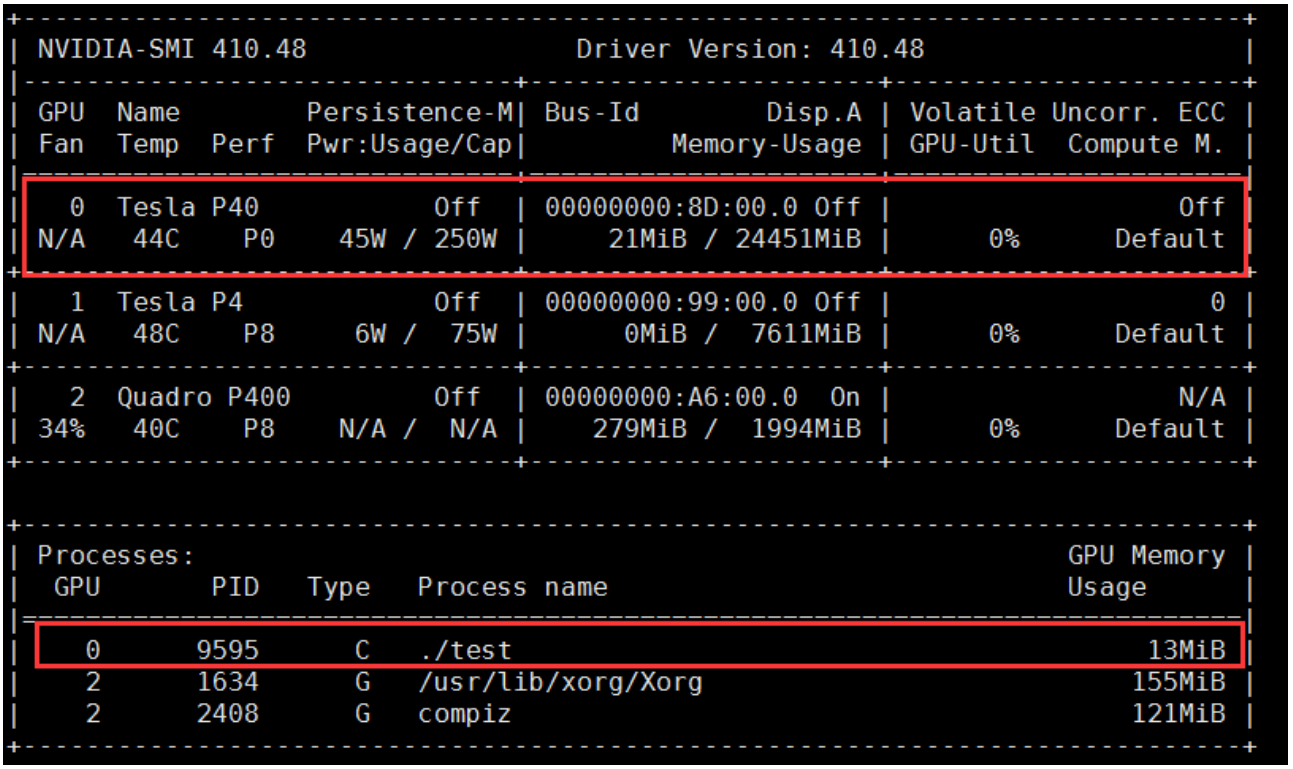


图4 指定了使用Telsa P40来计算

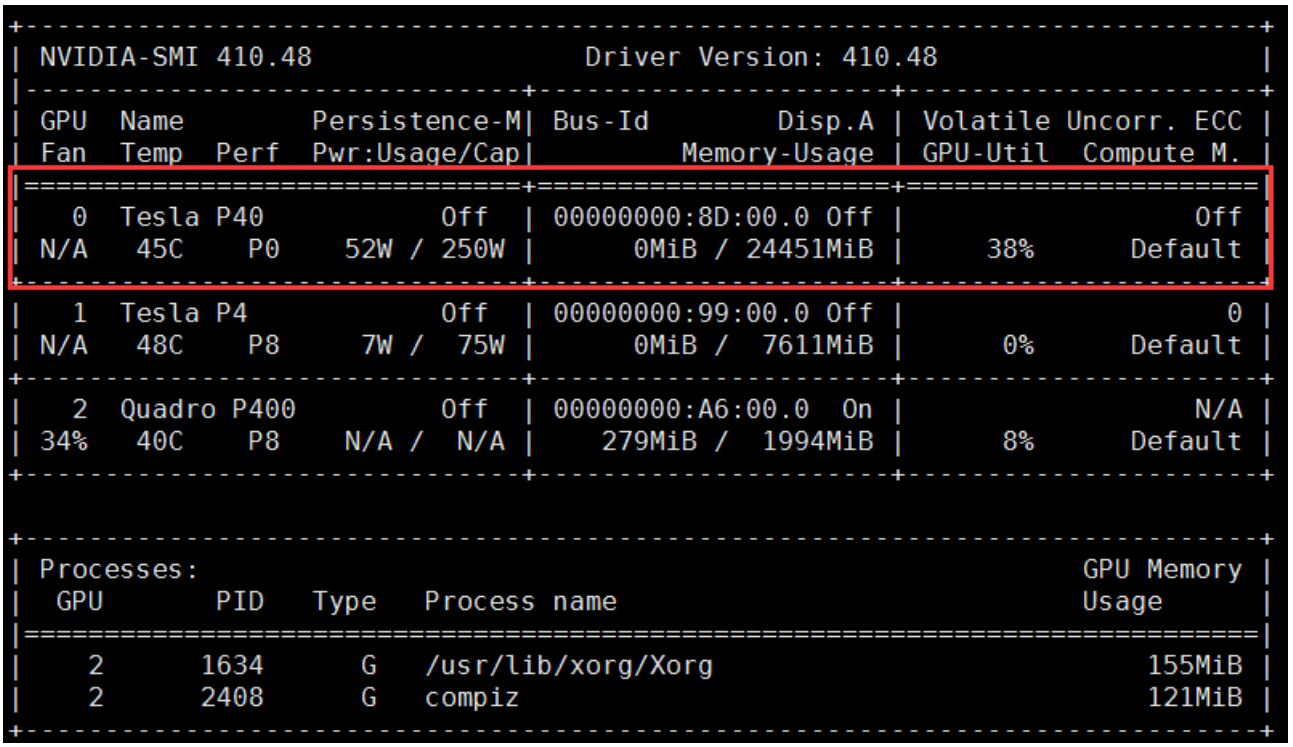


图5 指定了使用Telsa P40来计算

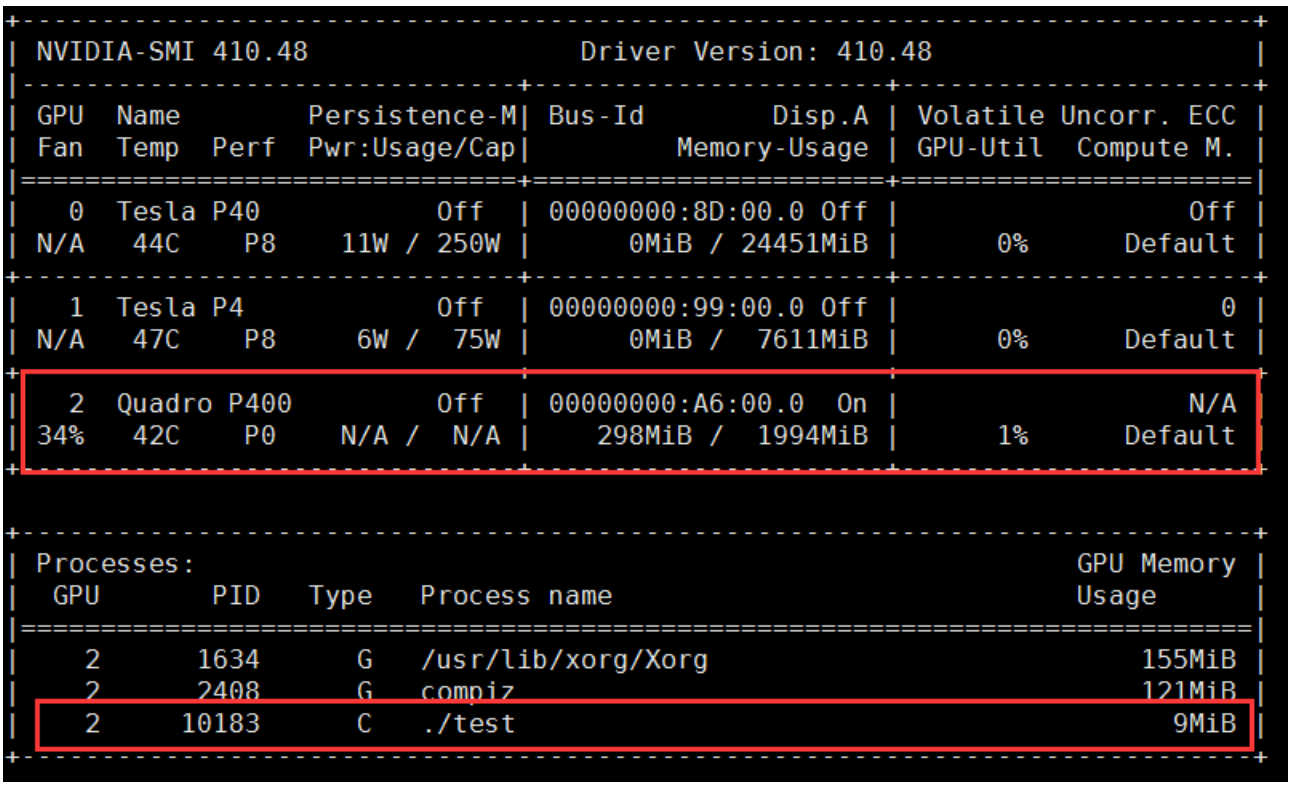


图5 指定了使用Quadro P400来计算

看到图中显示Telsa P40的Perf是P0，这代表该显卡达到最大性能。参数的具体解释为：

- 第一栏的Fan：N/A是风扇转速，从0到100%之间变动，这个速度是计算机期望的风扇转速，实际情况下如果风扇堵转，可能打不到显示的转速。有的设备不会返回转速，因为它不依赖风扇冷却而是通过其他外设保持低温（比如我们实验室的服务器是常年放在空调房间里的）。
- 第二栏的Temp：是温度，单位摄氏度。
- **第三栏的Perf：是性能状态，从P0到P12，P0表示最大性能，P12表示状态最小性能。**
- 第四栏下方的Pwr：是能耗，上方的Persistence-M：是持续模式的状态，持续模式虽然耗能大，但是在新的GPU应用启动时，花费的时间更少，这里显示的是off的状态。
- 第五栏的Bus-Id是涉及GPU总线的东西，domain: `bus` :device.function
- 第六栏的Disp.A是Display Active，表示GPU的显示是否初始化。
- 第五第六栏下方的Memory Usage是显存使用率。
- 第七栏是浮动的GPU利用率。
- 第八栏上方是关于ECC的东西。
- 第八栏下方Compute M是计算模式。

如何设置你想使用的GPU呢？只需要在环境变量(vim ~/.bashrc)中写入下面的信息：

```
export CUDA_VISIBLE_DEVICES="0"
```

或者在执行程序时使用下述命令：

```
CUDA_VISIBLE_DEVICES="0"  ./test     #在执行test文件时，使用device 0 GPU来执行
```

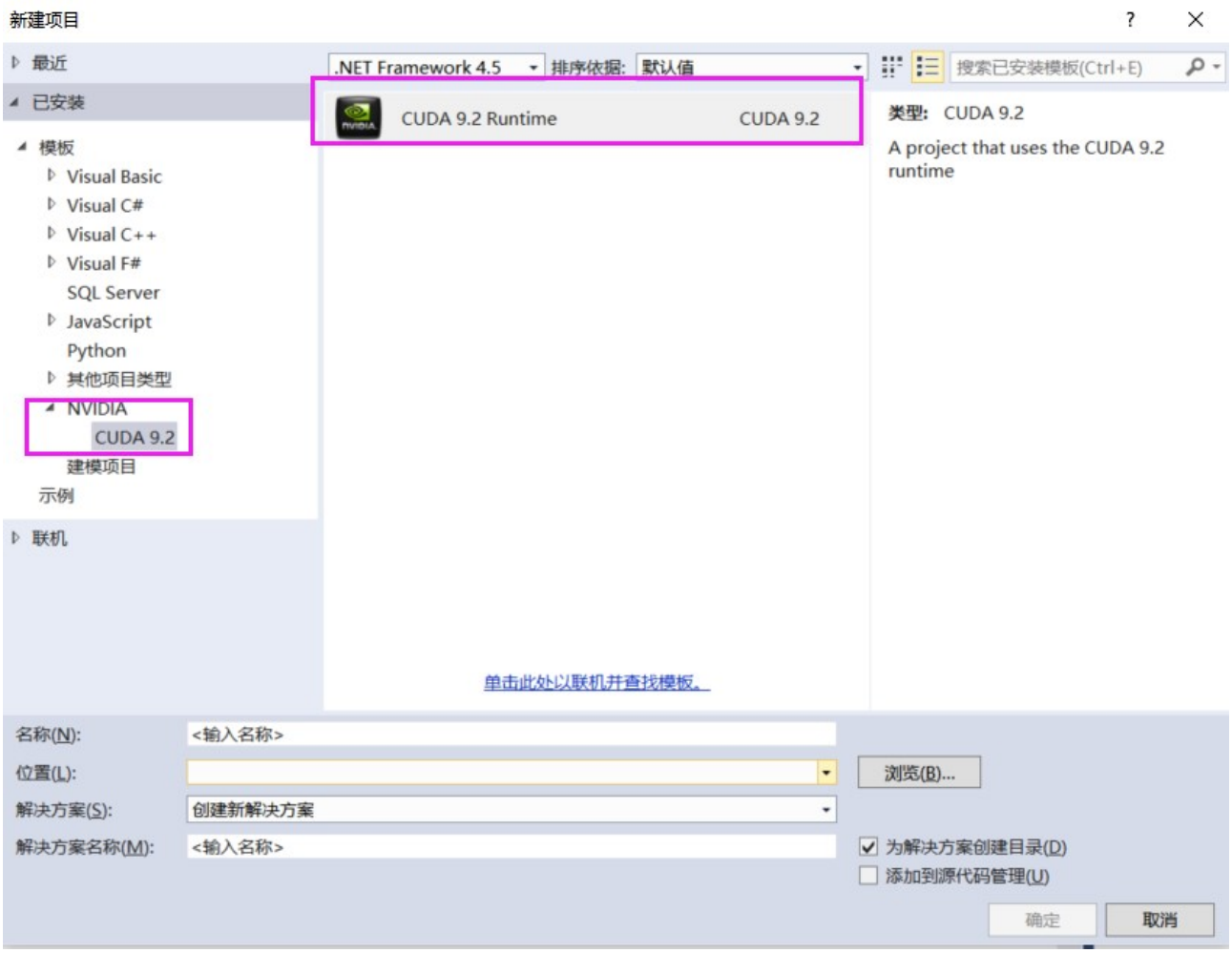
可执行文件在执行时即可看到指定的GPU的Perf参数为P0，也就是最大性能。

Visual Studio 2013与CUDA联合使用

如果你的计算机中成功安装了GPU驱动和CUDA，那么在Visual Studio 2013中就存在CUDA支持，相应的也就存在两种方式能够让你的计算机编译并执行CUDA代码，分别是自动配置和手动配置，如果你是新手，我推荐你使用自动配置，因为他极其简单，当然自动配置会生成大量无关的demo代码，如果你想手动配置，请直接参见手动配置部分。

自动配置

如图所示，打开Visual Studio 2013，接着新建项目，你会看到下面这样的界面，建立CUDA Runtime项目即可。



在自动配置中会生成下面的代码：

```
//代码实现了向量的并行加法，通过CUDA的线程模型实现
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

#include <stdio.h>
#include <stdlib.h>

cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size);

__global__ void addKernel(int *c, const int *a, const int *b)
{
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}

int main()
{
```

```
    const int arraySize = 5;
    const int a[arraySize] = { 1, 2, 3, 4, 5 };
    const int b[arraySize] = { 10, 20, 30, 40, 50 };
    int c[arraySize] = { 0 };

    // Add vectors in parallel.
    cudaError_t cudaStatus = addWithCuda(c, a, b, arraySize);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "addWithCuda failed!");
        return 1;
    }

    printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
        c[0], c[1], c[2], c[3], c[4]);

    // cudaDeviceReset must be called before exiting in order for profiling and
    // tracing tools such as Nsight and Visual Profiler to show complete traces.
    cudaStatus = cudaDeviceReset();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceReset failed!");
        return 1;
    }

    system("pause");
    return 0;
}

// Helper function for using CUDA to add vectors in parallel.
cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size)
{
    int *dev_a = 0;
    int *dev_b = 0;
    int *dev_c = 0;
    cudaError_t cudaStatus;

    // Choose which GPU to run on, change this on a multi-GPU system.
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
```

```
56          fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-
    capable GPU installed?");
57          goto Error;
58      }
59
60      // Allocate GPU buffers for three vectors (two input, one
    output)    .
61      cudaStatus = cudaMalloc((void**)&dev_c, size * sizeof(int));
62      if (cudaStatus != cudaSuccess) {
63          fprintf(stderr, "cudaMalloc failed!");
64          goto Error;
65      }
66
67      cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(int));
68      if (cudaStatus != cudaSuccess) {
69          fprintf(stderr, "cudaMalloc failed!");
70          goto Error;
71      }
72
73      cudaStatus = cudaMalloc((void**)&dev_b, size * sizeof(int));
74      if (cudaStatus != cudaSuccess) {
75          fprintf(stderr, "cudaMalloc failed!");
76          goto Error;
77      }
78
79      // Copy input vectors from host memory to GPU buffers.
80      cudaStatus = cudaMemcpy(dev_a, a, size * sizeof(int),
    cudaMemcpyHostToDevice);
81      if (cudaStatus != cudaSuccess) {
82          fprintf(stderr, "cudaMemcpy failed!");
83          goto Error;
84      }
85
86      cudaStatus = cudaMemcpy(dev_b, b, size * sizeof(int),
    cudaMemcpyHostToDevice);
87      if (cudaStatus != cudaSuccess) {
88          fprintf(stderr, "cudaMemcpy failed!");
89          goto Error;
90      }
91
92      // Launch a kernel on the GPU with one thread for each element.
93      addKernel<<<1, size>>>(dev_c, dev_a, dev_b);
```

```
94
95      // Check for any errors launching the kernel
96      cudaStatus = cudaGetLastError();
97      if (cudaStatus != cudaSuccess) {
98          fprintf(stderr, "addKernel launch failed: %s\n",
    cudaGetErrorString(cudaStatus));
99          goto Error;
100     }
101
102     // cudaDeviceSynchronize waits for the kernel to finish, and
    returns
103     // any errors encountered during the launch.
104     cudaStatus = cudaDeviceSynchronize();
105     if (cudaStatus != cudaSuccess) {
106         fprintf(stderr, "cudaDeviceSynchronize returned error code
    %d after launching addKernel!\n", cudaStatus);
107         goto Error;
108     }
109
110     // Copy output vector from GPU buffer to host memory.
111     cudaStatus = cudaMemcpy(c, dev_c, size * sizeof(int),
    cudaMemcpyDeviceToHost);
112     if (cudaStatus != cudaSuccess) {
113         fprintf(stderr, "cudaMemcpy failed!");
114         goto Error;
115     }
116
117 Error:
118     cudaFree(dev_c);
119     cudaFree(dev_a);
120     cudaFree(dev_b);
121
122     return cudaStatus;
123 }
```

通过修改上述的代码，可以实现任何的CUDA代码编译与运行。

手动配置

如果你是新手，推荐使用自动配置方式来新建工程后，在里面修改代码成为自己的工程，配置属性不会出错。若是进行手动配置也是可以的，只是过程非常繁琐，手动配

置过程参见： http://www.cnblogs.com/huliangwen/articles/5003680.html

参考资料

1. https://askubuntu.com/questions/700208/ubuntu-14-04-direct-rendering-not-working-with-nvidia
2. https://blog.csdn.net/A_Z666666/article/details/72853346
3. Python程序指定使用某个GPU执行程序 https://blog.csdn.net/ccbrid/article/details/79761674
4. 详细介绍了CUDA的初次使用步骤 https://blog.csdn.net/xsc_c/article/details/24250345