GPU Computing in Economics

Damian Clarke*

December 24, 2013

1 Introduction

CUDA is a series of programmes and utilities designed for parallel computing using the Graphics Processing Unit (GPU) available in nearly all new computers. This has been designed by NVIDIA for use with their GPUs. NVIDIA is a brand of GPUs used in many laptops, and is the market leader when considering dedicated units[1] which are suitable for parallel computation in economics (and many other computationally intensive fields).

In a nutshell, parallel computing allows for computationally intensive procedures to be separated and run in individual blocks rather than as one large job. This is particularly useful in applications such as Monte Carlo simulation, or other situations in which run many processes which are mutually independent are involved in arriving at a final result. Historically, parallel computation was run by splitting jobs and running them in unison over a small number of central processing units (or CPUs) which were available to a computer. However, with the advent of high powered GPUs for video games and other graphically-demanding jobs, many more cores were made available for running computations. For example, the most recent NVIDIA cards contain upwards of 1500 cores, each of which is capable of running an individual computation simultaneously.

Host: the CPU and its memory
Device: the GPU and its memory

*Contact: damian.clarke@economics.ox.ac.uk

[1]Often computers will have two GPUs; one "integrated" unit which is less power intensive and is used in every day tasks, and one "dedicated" unit which requires far more battery power, but which commands its own virtual memory and has much higher performance capabilities. It is these dedicated units which we focus on when undertaking parallel computing with the GPU.

Location of nvcc (compiler): /usr/local/cuda-5.5/bin/nvcc

A Installation

In this appendix I describe the process that I have followed in installing and running CUDA programs on an Optimus laptop running Ubuntu 12.04. The laptop I am installing this on has an NVIDIA GeForce GT 650M Graphics card with 2GB gDDR3 Graphic memory. The machine also has an integrated Intel Graphics card, hence the need for Optimus technology to run the dedicated NVIDIA card.

Optimus technology is designed to switch seamlessly between the dedicated GPU and integrated GPU when these both exist in the same machine. The idea of this is to both save power (when the dedicated GPU is not required), while taking advantage of the dedicated GPU when higher performance is necessary. However, this has created some difficulties in linux based operating systems given that many of the necessary drivers written by NVIDIA were not open source. This problem has been partially resolved by the Bumblebee Project which supports Optimus under Linux. In order to run CUDA, I ran a fresh install of Bumblebee's "Tumbleweed" release (version 3.2.1). In preparing to install CUDA, I first installed the most recent x-swat drivers, which bundle NVIDIA drivers for Xorg. This follows the advice provided on the following forum. and on Ubuntu looks like this:

```
sudo add-apt-repository ppa:ubuntu-x-swat/x-updates
sudo apt-get update
sudo apt-get upgrade
```

After installing the x-swat drivers, the current version of Bumblebee is installed:

```
sudo add-apt-repository ppa:bumblebee/stable
sudo apt-get update
sudo apt-get install bumblebee
```

At this point it is worth confirming that your Ubuntu system actually recognises the NVIDIA card with Optimus. Using `lspci` allows us to see all PCI devices in the system, and we are interested in the VGA video controller. On my system I confirmed that the NVIDIA card was recognised via:

```
damiancclarke@dcc-linux: $ lspci | grep VGA
```

```
00:02.0 VGA compatible controller:  Intel Corporation 3rd Gen Core processor Graphics Controller (rev 09)
01:00.0 VGA compatible controller:  NVIDIA Corporation GF108M [GeForce GT 630M] (rev ff)
```

It may also be worthwhile ensuring that Bumblebee is installed correctly, by referring to the installation instructions and tests described here. Now, assuming that Bumblebee is installed correctly, we can continue by downloading the CUDA Toolkit. The current version (at the time of this document) is 5.5, which offers a considerably smoother installation process than previous versions. I largely followed the instructions provided by NVIDIA in their Developer Zone, however given that this is mainly intended for non-Optimus machines, I outline the steps I followed precisely below[2]:

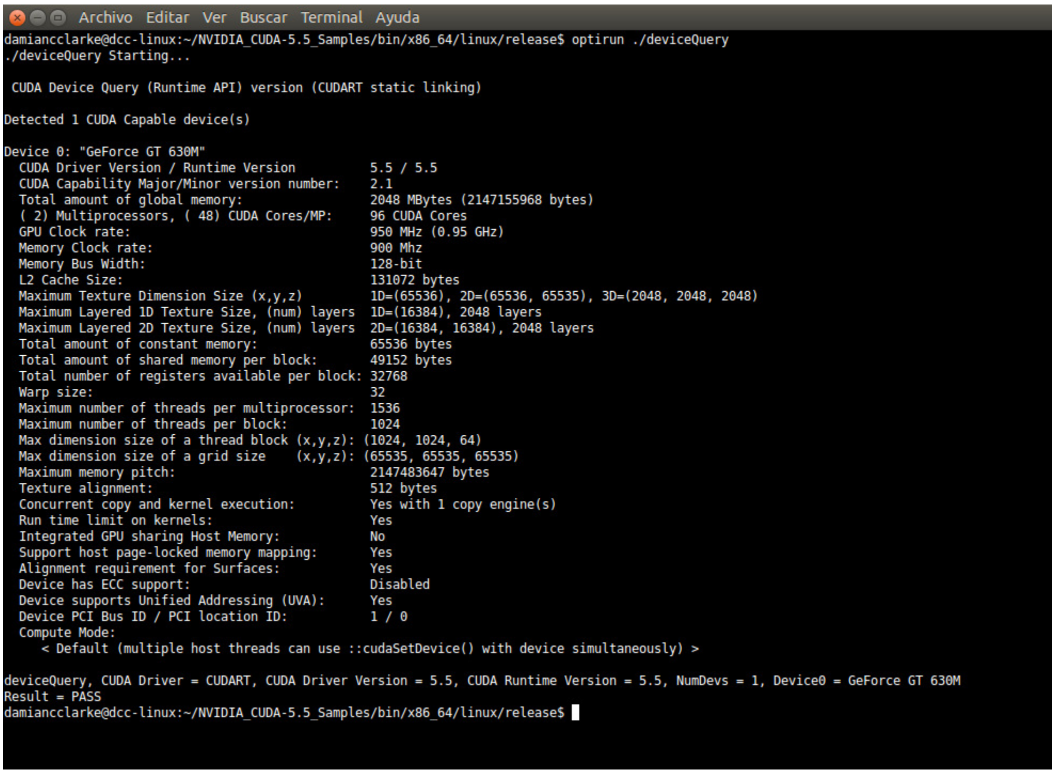
```
$ echo "foreign-architecture armhf" >> /etc/dpkg/dpkg.cfg.d/multiarch
$ sudo apt-get update
$ sudo dpkg -i cuda-repo-ubuntu1204_5.5-0_amd64.deb
$ sudo apt-get update
$ sudo apt-get install cuda
$ export PATH=/usr/local/cuda-5.5/bin:$PATH
$ export LD_LIBRARY_PATH=/usr/local/cuda-5.5/lib64:$LD_LIBRARY_PATH
```

In order to test that the above installation worked as desired, NVIDIA has provided a large number of sample programs. In order to compile and run these, I first changed to the CUDA installation directory (which in my case was " /usr/local/cuda-5.5/samples/") and then ran the following commands:

```
$ cuda-install-samples-5.5.sh   $ cd  /NVIDIA_CUDA-5.5_Samples/ $ make
$ cd bin/x86_64/linux/release $ optirun ./deviceQuery
```

It is this final line which offers the test regarding CUDA's functionality. If it has installed correctly, something like the following should be seen:

[2] A more comprehensive description is available in the aforementioned Developer Zone. Essentially I skipped certain steps, and slightly tweaked things by using Optirun, but the NVIDIA documentation is much more comprehensive to what I describe here.



```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
damianclarke@dcc-linux:~/NVIDIA_CUDA-5.5_Samples/bin/x86_64/linux/release$ optirun ./deviceQuery
./deviceQuery Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GT 630M"
  CUDA Driver Version / Runtime Version          5.5 / 5.5
  CUDA Capability Major/Minor version number:    2.1
  Total amount of global memory:                 2048 MBytes (2147155968 bytes)
  ( 2) Multiprocessors, ( 48) CUDA Cores/MP:     96 CUDA Cores
  GPU Clock rate:                                950 MHz (0.95 GHz)
  Memory Clock rate:                             900 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 131072 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65535), 3D=(2048, 2048, 2048)
  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:  1536
  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): (65535, 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
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Bus ID / PCI location ID:           1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 5.5, CUDA Runtime Version = 5.5, NumDevs = 1, Device0 = GeForce GT 630M
Result = PASS
damianclarke@dcc-linux:~/NVIDIA_CUDA-5.5_Samples/bin/x86_64/linux/release$
```