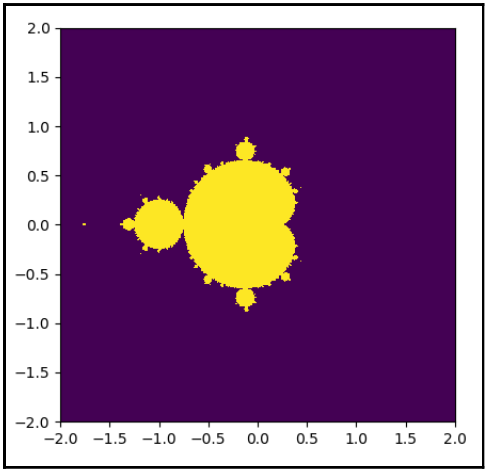
Chapter 1: Why GPU Programming?




```
PS C:\Users\btuom\examples\1> python mandelbrot0.py
It took 14.617000103 seconds to calculate the Mandelbrot graph.
It took 0.110999822617 seconds to dump the image.
```

```
PS C:\Users\btuom\examples\1> python -m cProfile -s cumtime mandelbrot0.py > mandelbrot_profile.txt
PS C:\Users\btuom\examples\1>
```

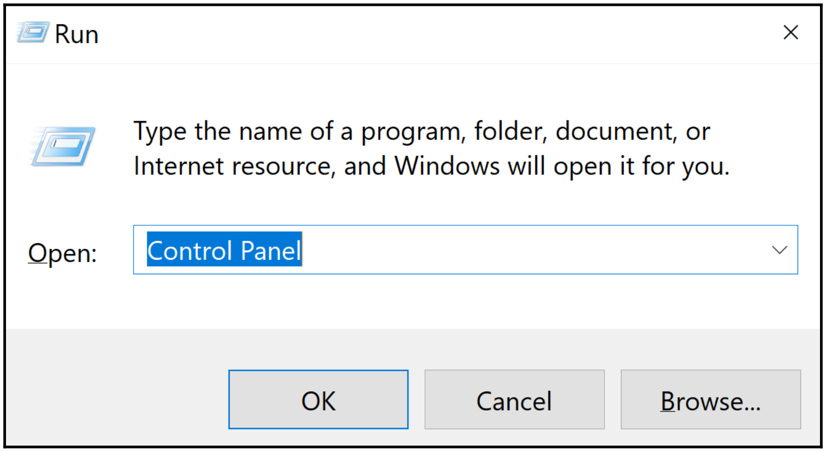
```
It took 14.5690000057 seconds to calculate the Mandelbrot graph.
It took 0.136000156403 seconds to dump the image.
         564104 function calls (559254 primitive calls) in 14.965 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.002    0.002   14.966   14.966 mandelbrot0.py:1(<module>)
        1   14.363   14.363   14.572   14.572 mandelbrot0.py:10(simple_mandelbrot)
   263606    0.209    0.000    0.209    0.000 {range}
        1    0.007    0.007    0.134    0.134 __init__.py:101(<module>)
        1    0.003    0.003    0.123    0.123 pyplot.py:17(<module>)
       12    0.017    0.001    0.119    0.010 __init__.py:1(<module>)
        1    0.000    0.000    0.097    0.097 pyplot.py:694(savefig)
        2    0.000    0.000    0.082    0.041 backend_agg.py:418(draw)
    152/2    0.000    0.000    0.081    0.041 artist.py:47(draw_wrapper)
        2    0.000    0.000    0.081    0.041 figure.py:1264(draw)
      4/2    0.000    0.000    0.080    0.040 image.py:120(_draw_list_compositing_images)
```

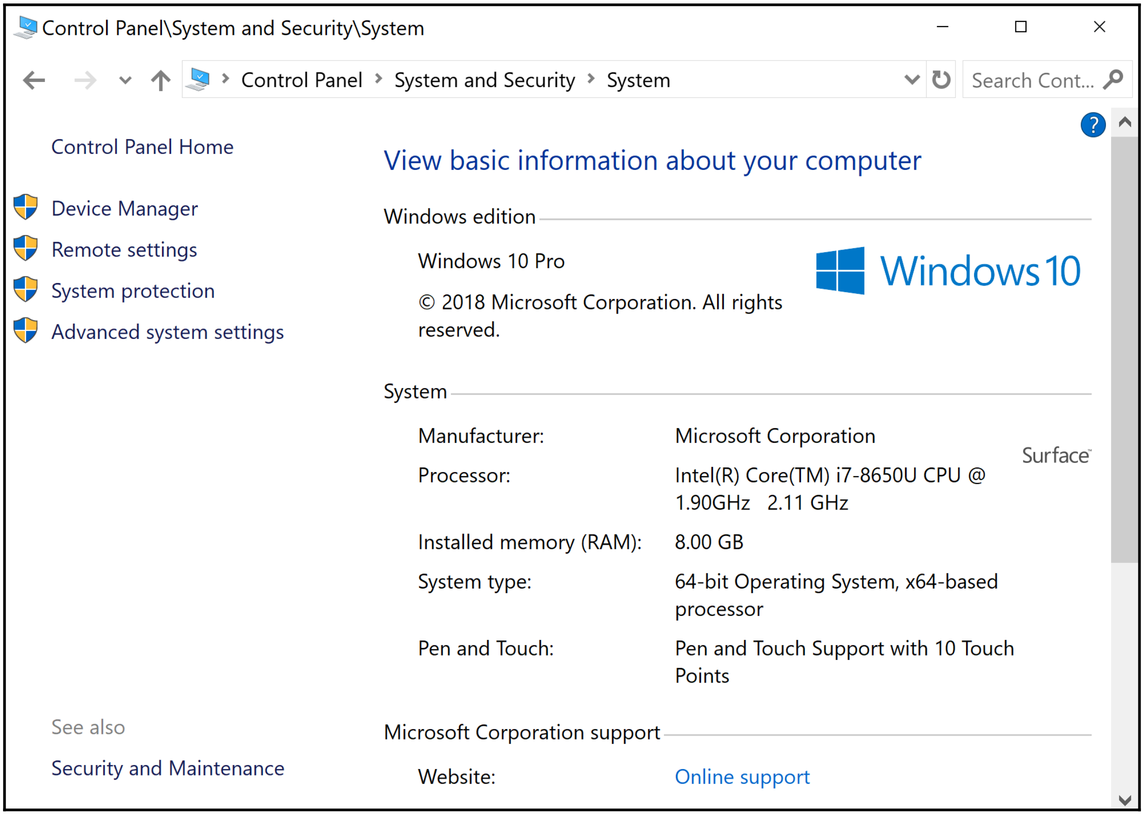
Chapter 2: Setting Up Your GPU Programming Environment

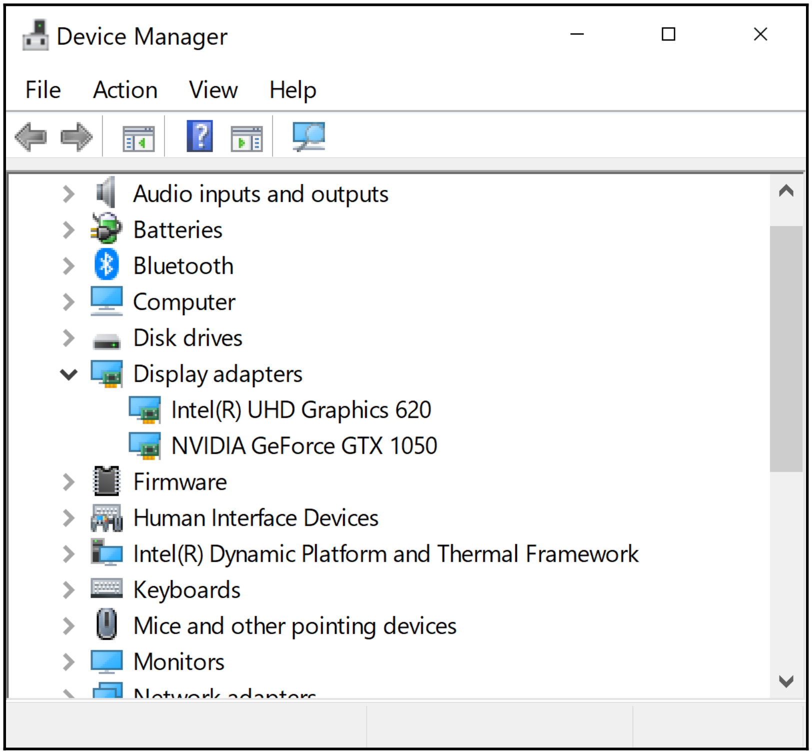
```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                12
On-line CPU(s) list:   0-11
Thread(s) per core:    2
Core(s) per socket:    6
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
```

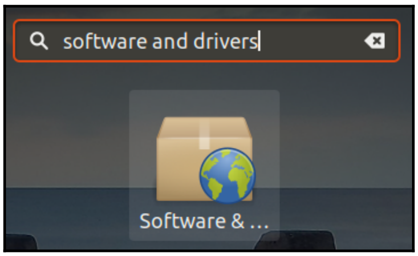
```
              total        used        free      shared  buff/cache   available
Mem:             15           3           9           0           2          12
Swap:             5           0           5
```

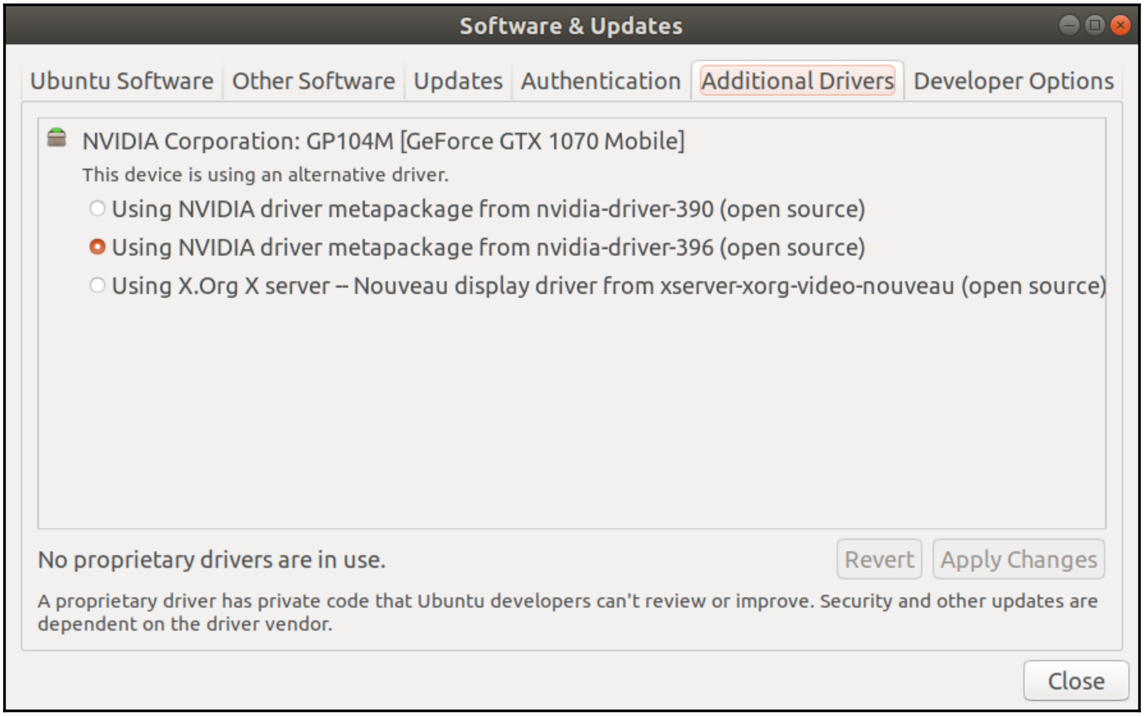
```
01:00.0 VGA compatible controller: NVIDIA Corporation GP104M [GeForce GTX 1070 Mobile] (rev a1)
```

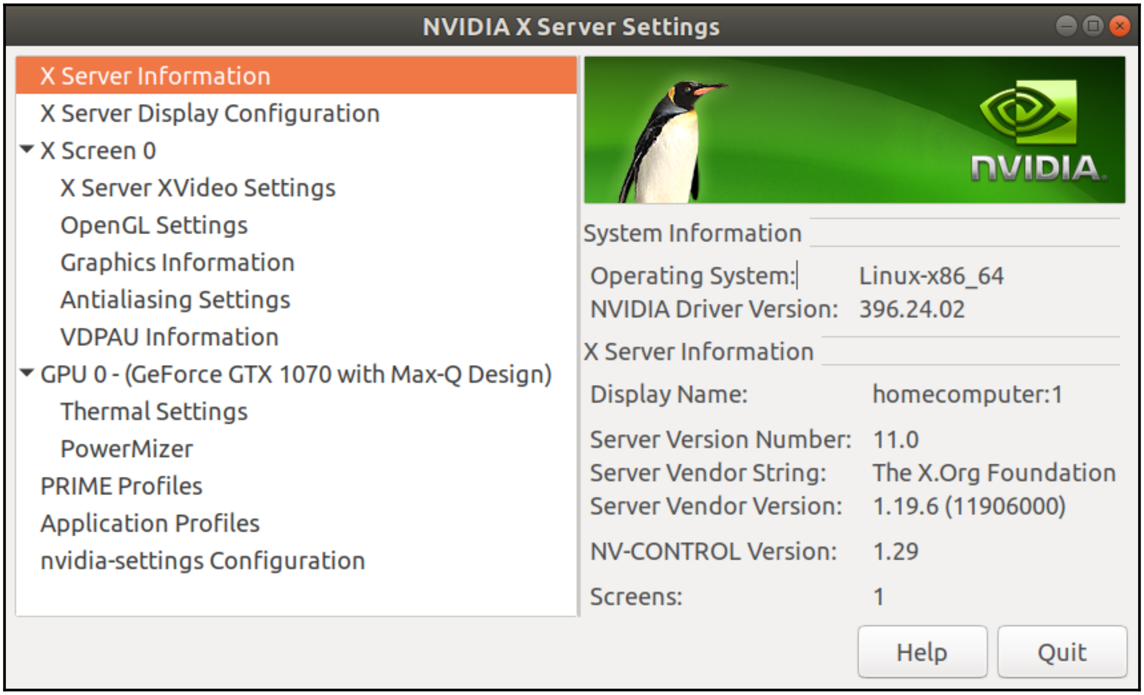


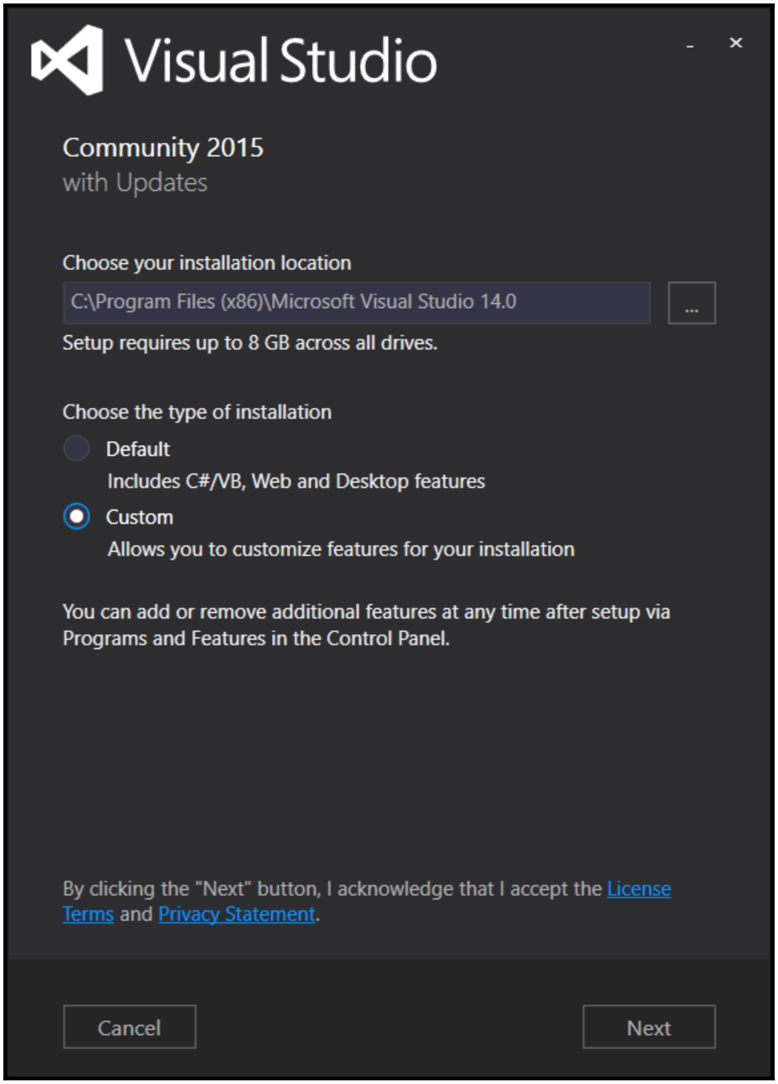

Control Panel\System and Security\System
Control Panel › System and Security › System
Search Cont...
Control Panel Home
Device Manager
Remote settings
System protection
Advanced system settings
See also
Security and Maintenance
View basic information about your computer
Windows edition
Windows 10 Pro
© 2018 Microsoft Corporation. All rights reserved.
Windows 10
System
Manufacturer:
Microsoft Corporation
Surface
Processor:
Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11 GHz
Installed memory (RAM):
8.00 GB
System type:
64-bit Operating System, x64-based processor
Pen and Touch:
Pen and Touch Support with 10 Touch Points
Microsoft Corporation support
Website:
Online support

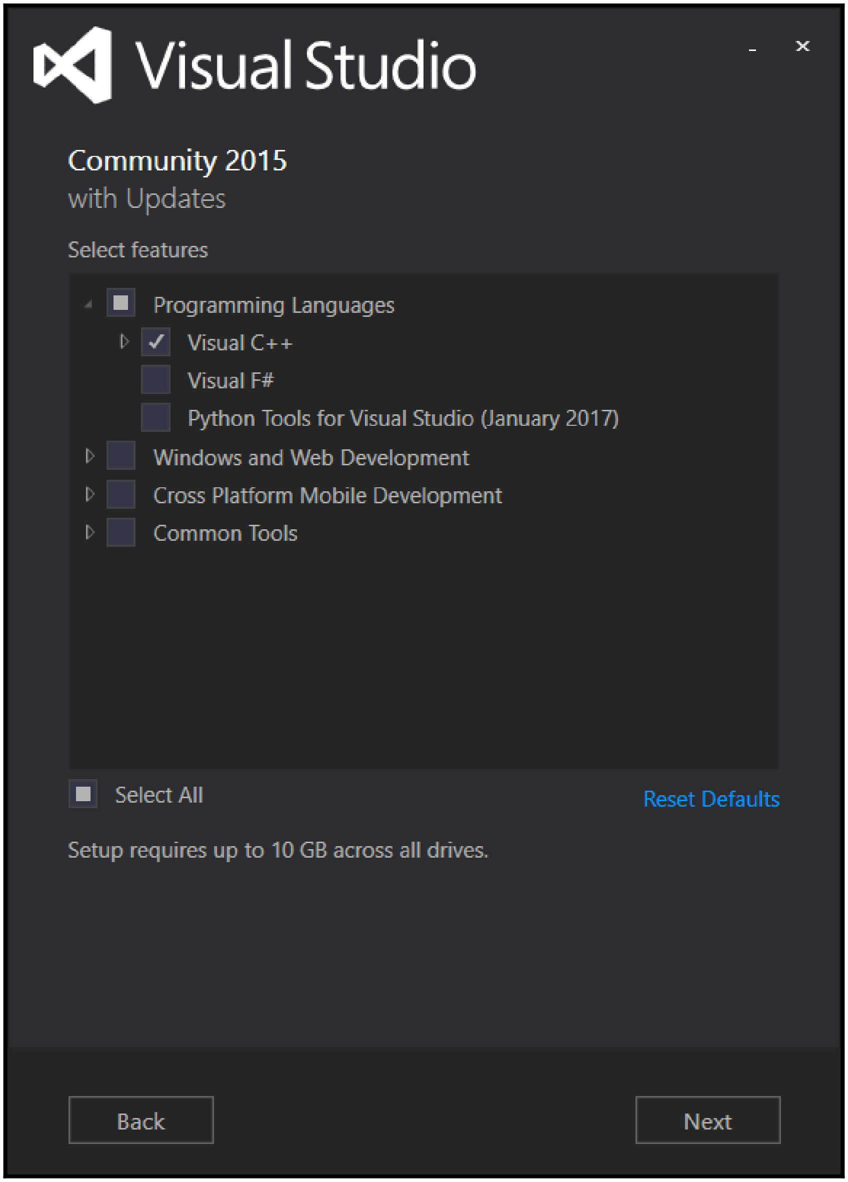

Device Manager
File
Action
View
Help
Audio inputs and outputs
Batteries
Bluetooth
Computer
Disk drives
Display adapters
Intel(R) UHD Graphics 620
NVIDIA GeForce GTX 1050
Firmware
Human Interface Devices
Intel(R) Dynamic Platform and Thermal Framework
Keyboards
Mice and other pointing devices
Monitors


software and drivers
Software & ...


Software & Updates
Ubuntu Software
Other Software
Updates
Authentication
Additional Drivers
Developer Options
NVIDIA Corporation: GP104M [GeForce GTX 1070 Mobile]
This device is using an alternative driver.
Using NVIDIA driver metapackage from nvidia-driver-390 (open source)
Using NVIDIA driver metapackage from nvidia-driver-396 (open source)
Using X.Org X server -- Nouveau display driver from xserver-xorg-video-nouveau (open source)
No proprietary drivers are in use.
Revert
Apply Changes
A proprietary driver has private code that Ubuntu developers can't review or improve. Security and other updates are dependent on the driver vendor.
Close


NVIDIA X Server Settings
X Server Information
X Server Display Configuration
X Screen 0
X Server XVideo Settings
OpenGL Settings
Graphics Information
Antialiasing Settings
VDPAU Information
GPU 0 - (GeForce GTX 1070 with Max-Q Design)
Thermal Settings
PowerMizer
PRIME Profiles
Application Profiles
nvidia-settings Configuration
NVIDIA
System Information
Operating System: Linux-x86_64
NVIDIA Driver Version: 396.24.02
X Server Information
Display Name: homecomputer:1
Server Version Number: 11.0
Server Vendor String: The X.Org Foundation
Server Vendor Version: 1.19.6 (11906000)
NV-CONTROL Version: 1.29
Screens: 1
Help
Quit


Visual Studio
Community 2015
with Updates
Choose your installation location
C:\Program Files (x86)\Microsoft Visual Studio 14.0
...
Setup requires up to 8 GB across all drives.
Choose the type of installation
Default
Includes C#/VB, Web and Desktop features
Custom
Allows you to customize features for your installation
You can add or remove additional features at any time after setup via Programs and Features in the Control Panel.
By clicking the "Next" button, I acknowledge that I accept the License Terms and Privacy Statement.
Cancel
Next




```
PS C:\Users\btuom\examples\3> python .\deviceQuery.py
CUDA device query (PyCUDA version)

Detected 1 CUDA Capable device(s)

Device 0: GeForce GTX 1050
```

```
PS C:\ProgramData\NVIDIA Corporation\CUDA Samples\v9.1\bin\win64\Debug> .\deviceQuery.exe
C:\ProgramData\NVIDIA Corporation\CUDA Samples\v9.1\bin\win64\Debug\deviceQuery.exe Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1050"
  CUDA Driver Version / Runtime Version          9.1 / 9.1
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 2048 MBytes (2147483648 bytes)
  ( 5) Multiprocessors, (128) CUDA Cores/MP:     640 CUDA Cores
  GPU Max Clock rate:                            1493 MHz (1.49 GHz)
  Memory Clock rate:                             3504 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 524288 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:                     No
  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 / 2 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.1, CUDA Runtime Version = 9.1, NumDevs = 1
Result = PASS
```

```
In [8]: import pycuda.driver as drv

In [9]: drv.init()

In [10]: print 'Detected {} CUDA Capable device(s)'.format(drv.Device.count())
Detected 1 CUDA Capable device(s)
```

```
PS C:\Users\btuom\examples\3> python deviceQuery.py
CUDA device query (PyCUDA version)

Detected 1 CUDA Capable device(s)

Device 0: GeForce GTX 1050
         Compute Capability: 6.1
         Total Memory: 2048 megabytes
         (5) Multiprocessors, (128) CUDA Cores / Multiprocessor: 640 CUDA Cores
         MAXIMUM_TEXTURE2D_LINEAR_PITCH: 2097120
         MAXIMUM_TEXTURE2D_GATHER_WIDTH: 32768
         MAXIMUM_TEXTURE2D_GATHER_HEIGHT: 32768
         PCI_DEVICE_ID: 0
         MAXIMUM_TEXTURE3D_WIDTH: 16384
         MAXIMUM_SURFACE2D_WIDTH: 131072
         MAXIMUM_TEXTURE1D_MIPMAPPED_WIDTH: 16384
         GLOBAL_MEMORY_BUS_WIDTH: 128
         LOCAL_L1_CACHE_SUPPORTED: 1
         MAXIMUM_SURFACE3D_DEPTH: 16384
         MAXIMUM_TEXTURE3D_HEIGHT: 16384
         PCI_DOMAIN_ID: 0
         COMPUTE_CAPABILITY_MINOR: 1
         MULTI_GPU_BOARD_GROUP_ID: 0
         MAX_REGISTERS_PER_BLOCK: 65536
         MAXIMUM_TEXTURE2D_ARRAY_WIDTH: 32768
         COMPUTE_CAPABILITY_MAJOR: 6
         MAXIMUM_SURFACE2D_LAYERED_HEIGHT: 32768
         MAXIMUM_TEXTURE1D_LAYERED_LAYERS: 2048
         UNIFIED_ADDRESSING: 1
```

```
In [24]: import numpy as np

In [25]: import pycuda.autoinit

In [26]: from pycuda import gpuarray

In [27]: host_data = np.array([1,2,3,4,5],dtype=np.float32)

In [28]: device_data = gpuarray.to_gpu(host_data)

In [29]: device_data_x2 = 2 * device_data

In [30]: host_data_x2 = device_data_x2.get()

In [31]: print host_data_x2
[  2.   4.   6.   8.  10.]

In [32]:
```

```
In [14]: x_host = np.array([1,2,3], dtype=np.float32)

In [15]: y_host = np.array([1,1,1], dtype=np.float32)

In [16]: z_host = np.array([2,2,2], dtype=np.float32)

In [17]: x_device = gpuarray.to_gpu(x_host)

In [18]: y_device = gpuarray.to_gpu(y_host)

In [19]: z_device = gpuarray.to_gpu(z_host)

In [20]: x_host + y_host
Out[20]: array([ 2.,  3.,  4.], dtype=float32)

In [21]: (x_device + y_device).get()
Out[21]: array([ 2.,  3.,  4.], dtype=float32)

In [22]: x_host ** z_host
Out[22]: array([ 1.,  4.,  9.], dtype=float32)

In [23]: (x_device ** z_device).get()
Out[23]: array([ 1.,  4.,  9.], dtype=float32)

In [24]: x_host / x_host
Out[24]: array([ 1.,  1.,  1.], dtype=float32)

In [25]: (x_device / x_device).get()
Out[25]: array([ 1.,  1.,  1.], dtype=float32)

In [26]: z_host - x_host
Out[26]: array([ 1.,  0., -1.], dtype=float32)

In [27]: (z_device - x_device).get()
Out[27]: array([ 1.,  0., -1.], dtype=float32)

In [28]: z_host / 2
Out[28]: array([ 1.,  1.,  1.], dtype=float32)

In [29]: (z_device / 2).get()
Out[29]: array([ 1.,  1.,  1.], dtype=float32)

In [30]: x_host - 1
Out[30]: array([ 0.,  1.,  2.], dtype=float32)

In [31]: (x_device - 1).get()
Out[31]: array([ 0.,  1.,  2.], dtype=float32)
```

```
In [1]: run time_calc0.py
total time to compute on CPU: 0.078000
total time to compute on GPU: 1.094000
Is the host computation the same as the GPU computation? : True

In [2]: run time_calc0.py
total time to compute on CPU: 0.079000
total time to compute on GPU: 0.008000
Is the host computation the same as the GPU computation? : True

In [3]: run time_calc0.py
total time to compute on CPU: 0.080000
total time to compute on GPU: 0.007000
Is the host computation the same as the GPU computation? : True

In [4]: run time_calc0.py
total time to compute on CPU: 0.078000
total time to compute on GPU: 0.009000
Is the host computation the same as the GPU computation? : True

In [5]: run time_calc0.py
total time to compute on CPU: 0.079000
total time to compute on GPU: 0.009000
Is the host computation the same as the GPU computation? : True
```

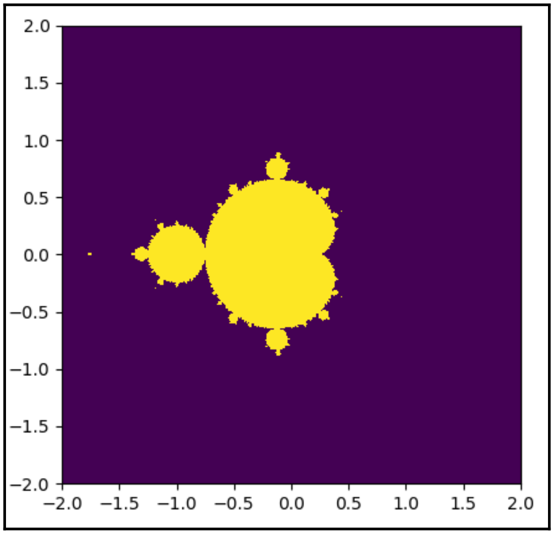
```
In [2]: %prun -s cumulative exec(time_calc_code)
total time to compute on CPU: 0.078000
total time to compute on GPU: 1.100000
Is the host computation the same as the GPU computation? : True
         17353 function calls (17146 primitive calls) in 3.175 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    1.101    1.101 gpuarray.py:452(__mul__)
        1    0.000    0.000    1.092    1.092 gpuarray.py:317(_axpbz)
        1    0.000    0.000    1.091    1.091 <decorator-gen-122>:1(get_axpbz_kernel)

        1    0.000    0.000    1.091    1.091 tools.py:414(context_dependent_memoize)

        1    0.000    0.000    1.091    1.091 elementwise.py:413(get_axpbz_kernel)
        1    0.000    0.000    1.091    1.091 elementwise.py:155(get_elwise_kernel)
        1    0.000    0.000    1.091    1.091 elementwise.py:126(get_elwise_kernel_an
d_types)
        1    0.000    0.000    1.091    1.091 elementwise.py:41(get_elwise_module)
        1    0.001    0.001    1.089    1.089 compiler.py:285(__init__)
        1    0.001    0.001    1.089    1.089 compiler.py:190(compile)
        1    0.001    0.001    1.070    1.070 compiler.py:69(compile_plain)
        2    0.000    0.000    1.061    0.531 prefork.py:222(call_capture_output)
        2    0.000    0.000    1.061    0.531 prefork.py:43(call_capture_output)
        1    0.000    0.000    0.950    0.950 compiler.py:36(preprocess_source)
        2    0.000    0.000    0.837    0.419 subprocess.py:448(communicate)
        2    0.000    0.000    0.837    0.419 subprocess.py:698(_communicate)
        6    0.000    0.000    0.836    0.139 threading.py:309(wait)
```

```
In [3]: %prun -s cumulative exec(time_calc_code)
total time to compute on CPU: 0.101000
total time to compute on GPU: 0.015000
Is the host computation the same as the GPU computation? : True
         342 function calls (336 primitive calls) in 1.315 seconds

   Ordered by: cumulative time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    1.606    1.606 <string>:1(<module>)
        1    0.016    0.016    0.650    0.650 numeric.py:2397(allclose)
        1    0.069    0.069    0.630    0.630 numeric.py:2463(isclose)
        1    0.400    0.400    0.554    0.554 numeric.py:2522(within_tol)
        1    0.452    0.452    0.452    0.452 {method 'random_sample' of 'mtrand.Rand
omState' objects}
        2    0.191    0.096    0.191    0.096 gpuarray.py:1174(_memcpy_discontig)
        2    0.154    0.077    0.154    0.077 {abs}
        1    0.000    0.000    0.107    0.107 gpuarray.py:248(get)
        1    0.000    0.000    0.094    0.094 gpuarray.py:990(to_gpu)
        1    0.000    0.000    0.085    0.085 gpuarray.py:230(set)
        2    0.018    0.009    0.018    0.009 gpuarray.py:162(__init__)
        3    0.000    0.000    0.012    0.004 fromnumeric.py:1973(all)
```

```
PS C:\Users\btuom\examples\3> python simple_element_kernel_example0.py
total time to compute on CPU: 0.092000
total time to compute on GPU: 1.494000
Is the host computation the same as the GPU computation? : True
PS C:\Users\btuom\examples\3>
```

```
In [1]: run simple_element_kernel_example0.py
total time to compute on CPU: 0.080000
total time to compute on GPU: 0.989000
Is the host computation the same as the GPU computation? : True

In [2]: speedcomparison()
total time to compute on CPU: 0.081000
total time to compute on GPU: 0.000000
Is the host computation the same as the GPU computation? : True

In [3]: speedcomparison()
total time to compute on CPU: 0.096000
total time to compute on GPU: 0.000000
Is the host computation the same as the GPU computation? : True

In [4]: speedcomparison()
total time to compute on CPU: 0.085000
total time to compute on GPU: 0.000000
Is the host computation the same as the GPU computation? : True

In [5]: speedcomparison()
total time to compute on CPU: 0.085000
total time to compute on GPU: 0.000000
Is the host computation the same as the GPU computation? : True

In [6]:
```

```
In [1]: run gpu_mandelbrot0.py
It took 0.894000053406 seconds to calculate the Mandelbrot graph.
It took 0.102999925613 seconds to dump the image.
```

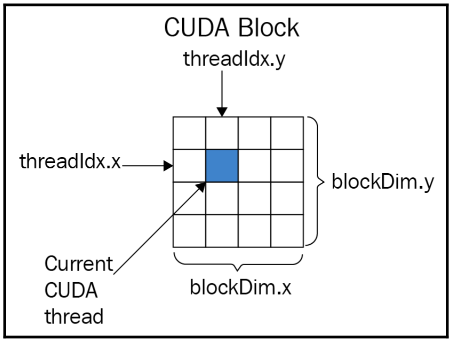


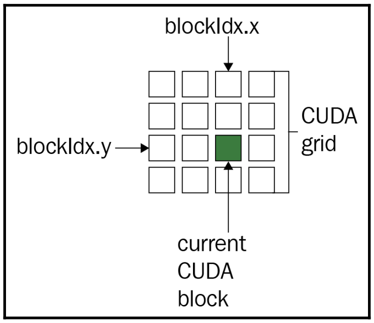

```
In [2]: pow2 = lambda x : x**2

In [3]: pow2(2)
Out[3]: 4

In [4]: pow2(3)
Out[4]: 9

In [5]: pow2(4)
Out[5]: 16
```

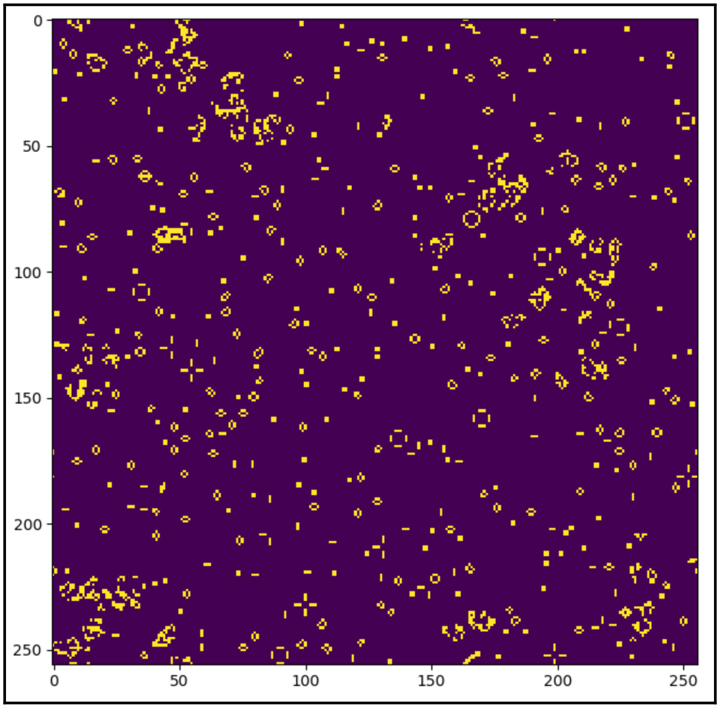
```
In [6]: map(lambda x : x**2, [2,3,4])
Out[6]: [4, 9, 16]
```

```
In [1]: run simple_scankernel0.py
[ 1  3  6 10]
[ 1  3  6 10]
```

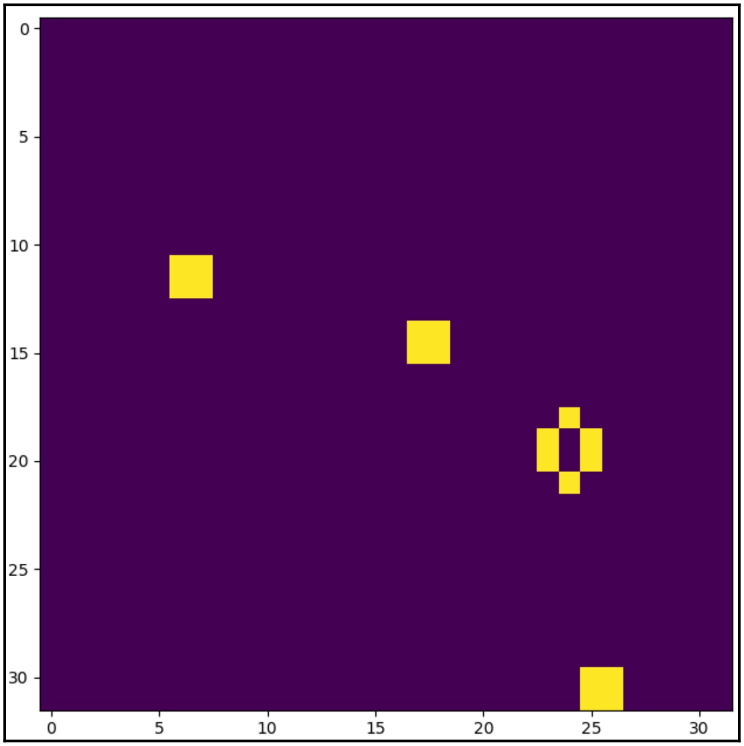
Chapter 4: Kernels, Threads, Blocks, and Grids







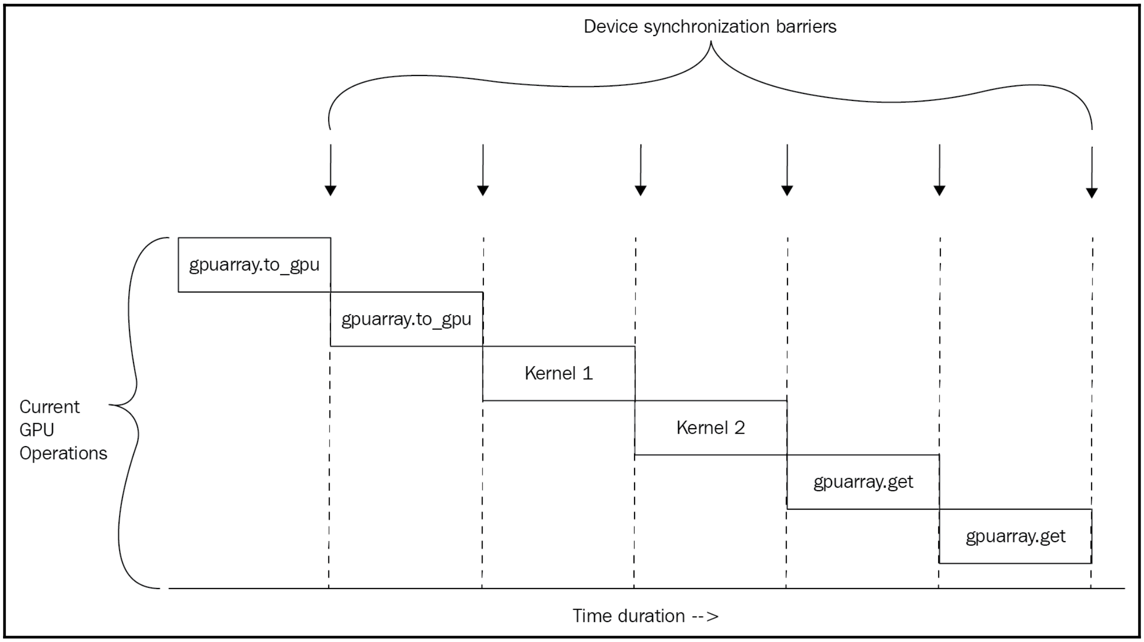
0
50
100
150
200
250
0
50
100
150
200
250

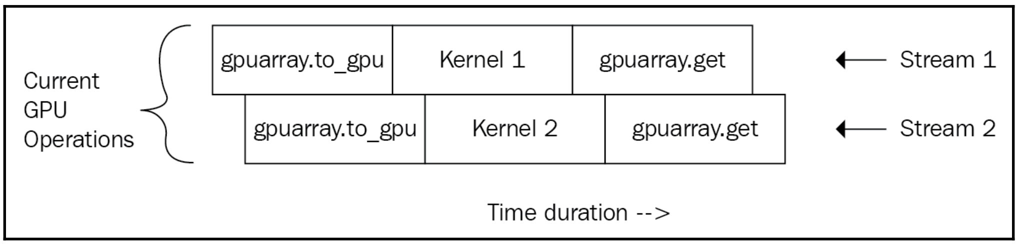

0
5
10
15
20
25
30
0
5
10
15
20
25
30

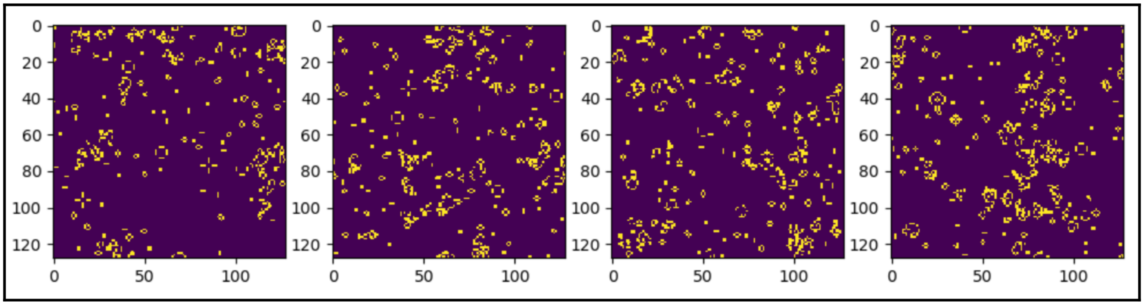
Chapter 5: Streams, Events, Contexts, and Concurrency

```
PS C:\Users\btuom\examples\5> python .\multi-kernel.py
Total time: 2.976000
```

```
PS C:\Users\btuom\examples\5> python .\multi-kernel_streams.py
Total time: 0.945000
```








```
PS C:\Users\btuom\examples\5> python .\simple_event_example.py
Has the kernel started yet? False
Has the kernel ended yet? False
```

```
PS C:\Users\btuom\examples\5> python .\simple_event_example.py
Has the kernel started yet? True
Has the kernel ended yet? True
Kernel execution time in milliseconds: 1047.391235
```

```
PS C:\Users\btuom\examples\5> python .\multi-kernel_events.py
Total time: 1.078000
Mean kernel duration (milliseconds): 71.417903
Mean kernel standard deviation (milliseconds): 6.401030
```

```
PS C:\Users\btuom\examples\5> python .\single_thread_example.py
Hello from the thread you just spawned!
The thread completed and returned this value: 123
```

Chapter 6: Debugging and Profiling Your CUDA Code

```
PS C:\Users\btuom\examples\6> python .\hello-world_gpu.py
Hello world from thread 0, in block 1!
Hello world from thread 1, in block 1!
Hello world from thread 2, in block 1!
Hello world from thread 3, in block 1!
Hello world from thread 4, in block 1!
Hello world from thread 0, in block 0!
Hello world from thread 1, in block 0!
Hello world from thread 2, in block 0!
Hello world from thread 3, in block 0!
Hello world from thread 4, in block 0!
-------------------------------------
This kernel was launched over a grid consisting of 2 blocks,
where each block has 5 threads.
```

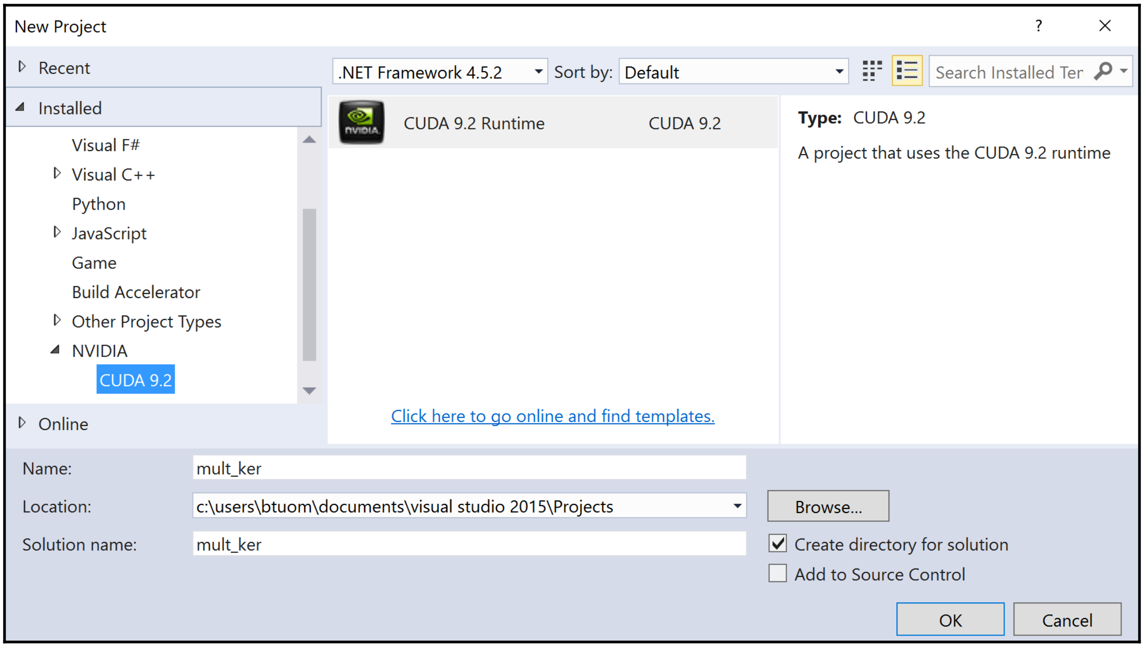
```
PS C:\Users\btuom\examples\6> python .\broken_matrix_ker.py
Traceback (most recent call last):
  File ".\broken_matrix_ker.py", line 64, in <module>
    assert( np.allclose(output_mat_gpu.get(), output_mat) )
AssertionError
PS C:\Users\btuom\examples\6>
```

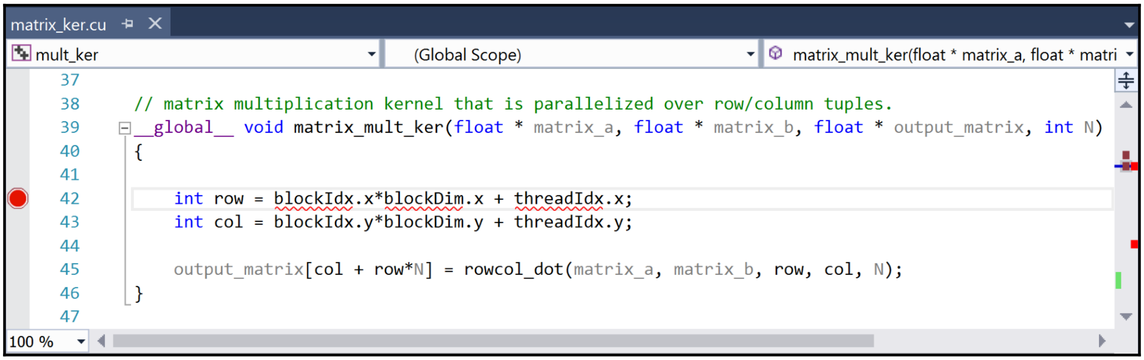
```
PS C:\Users\btuom\examples\6> python .\broken_matrix_ker.py
threadIdx.x,y: 0,0 blockIdx.x,y: 1,0 -- row is 1, col is 0.
threadIdx.x,y: 1,0 blockIdx.x,y: 1,0 -- row is 2, col is 0.
threadIdx.x,y: 0,1 blockIdx.x,y: 1,0 -- row is 1, col is 1.
threadIdx.x,y: 1,1 blockIdx.x,y: 1,0 -- row is 2, col is 1.
threadIdx.x,y: 0,0 blockIdx.x,y: 1,1 -- row is 1, col is 1.
threadIdx.x,y: 1,0 blockIdx.x,y: 1,1 -- row is 2, col is 1.
threadIdx.x,y: 0,1 blockIdx.x,y: 1,1 -- row is 1, col is 2.
threadIdx.x,y: 1,1 blockIdx.x,y: 1,1 -- row is 2, col is 2.
threadIdx.x,y: 0,0 blockIdx.x,y: 0,0 -- row is 0, col is 0.
threadIdx.x,y: 1,0 blockIdx.x,y: 0,0 -- row is 1, col is 0.
threadIdx.x,y: 0,1 blockIdx.x,y: 0,0 -- row is 0, col is 1.
threadIdx.x,y: 1,1 blockIdx.x,y: 0,0 -- row is 1, col is 1.
threadIdx.x,y: 0,0 blockIdx.x,y: 0,1 -- row is 0, col is 1.
threadIdx.x,y: 1,0 blockIdx.x,y: 0,1 -- row is 1, col is 1.
threadIdx.x,y: 0,1 blockIdx.x,y: 0,1 -- row is 0, col is 2.
threadIdx.x,y: 1,1 blockIdx.x,y: 0,1 -- row is 1, col is 2.
Traceback (most recent call last):
  File ".\broken_matrix_ker.py", line 64, in <module>
    assert( np.allclose(output_mat_gpu.get(), output_mat) )
AssertionError
PS C:\Users\btuom\examples\6>
```

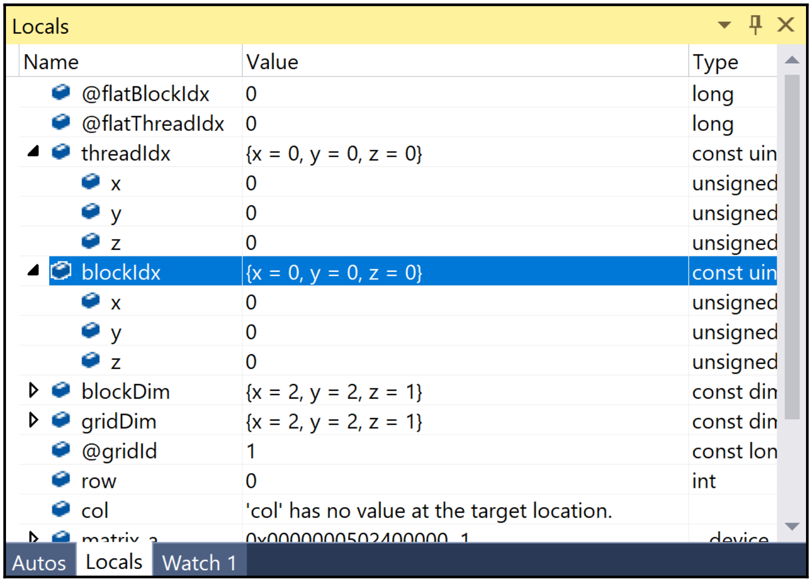
```
In [2]: print test_a
[[ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]
 [ 1.  2.  3.  4.]]

In [3]: print test_b
[[ 14.  13.  12.  11.]
 [ 14.  13.  12.  11.]
 [ 14.  13.  12.  11.]
 [ 14.  13.  12.  11.]]

In [4]:
```

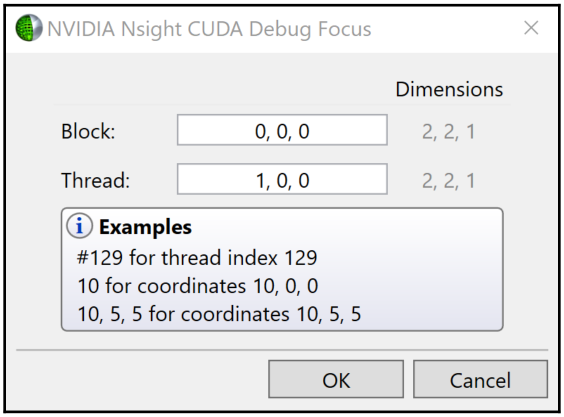
```
Dot-product loop: k value is 0, matrix_a value is 1.000000, matrix_b is 14.000000.
Dot-product loop: k value is 1, matrix_a value is 1.000000, matrix_b is 13.000000.
Dot-product loop: k value is 2, matrix_a value is 1.000000, matrix_b is 12.000000.
Dot-product loop: k value is 3, matrix_a value is 1.000000, matrix_b is 11.000000.
```

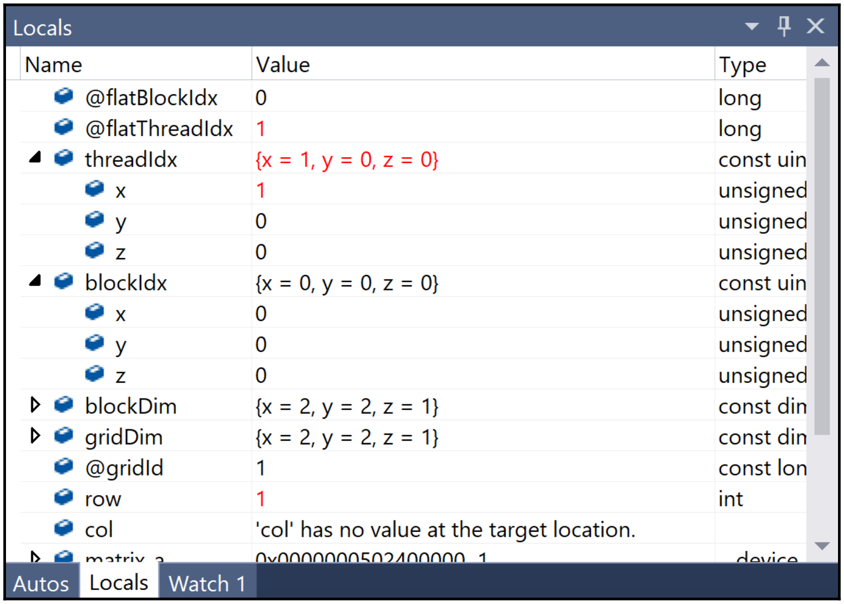
```
PS C:\Users\btuom\examples\6> nvcc matrix_ker.cu -o matrix_ker
matrix_ker.cu
   Creating library matrix_ker.lib and object matrix_ker.exp
PS C:\Users\btuom\examples\6> .\matrix_ker.exe
Success!  Output of kernel matches expected output.
PS C:\Users\btuom\examples\6>
```

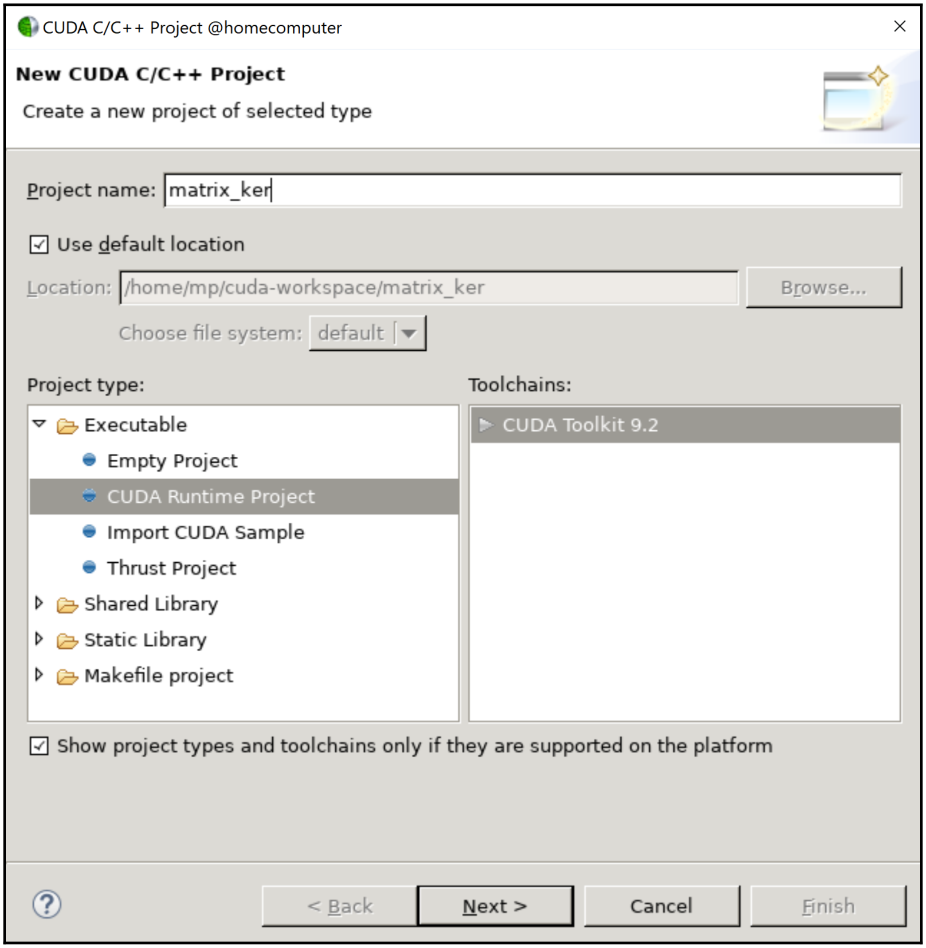


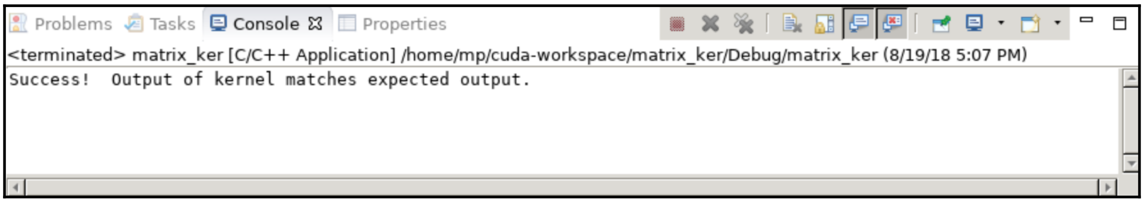


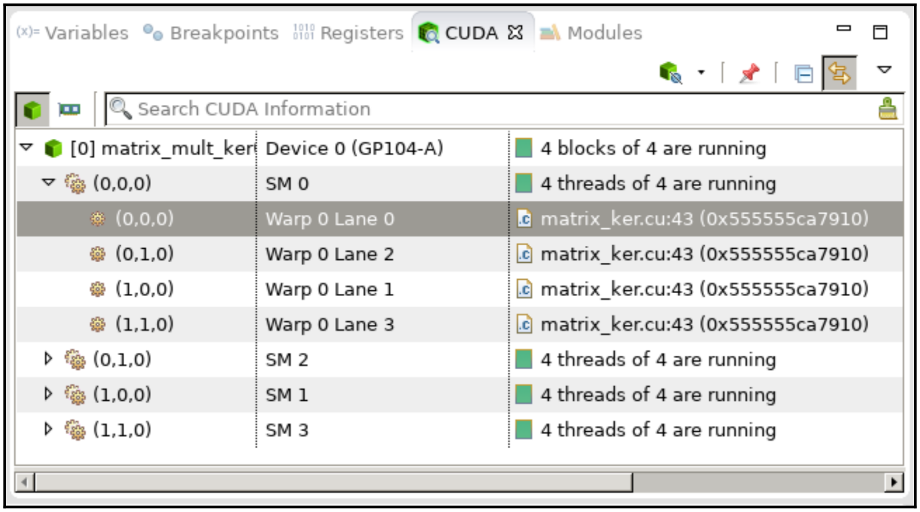

Locals
Name
Value
Type
@flatBlockIdx
0
long
@flatThreadIdx
0
long
threadIdx
{x = 0, y = 0, z = 0}
const uin
x
0
unsigned
y
0
unsigned
z
0
unsigned
blockIdx
{x = 0, y = 0, z = 0}
const uin
x
0
unsigned
y
0
unsigned
z
0
unsigned
blockDim
{x = 2, y = 2, z = 1}
const din
gridDim
{x = 2, y = 2, z = 1}
const din
@gridId
1
const lon
row
0
int
col
'col' has no value at the target location.
Autos
Locals
Watch 1

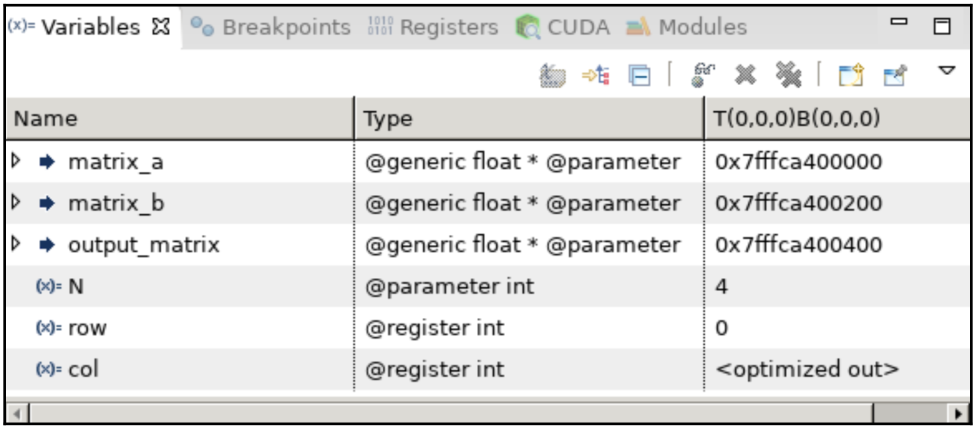

NVIDIA Nsight CUDA Debug Focus
Dimensions
Block:
0, 0, 0
2, 2, 1
Thread:
1, 0, 0
2, 2, 1
Examples
#129 for thread index 129
10 for coordinates 10, 0, 0
10, 5, 5 for coordinates 10, 5, 5
OK
Cancel

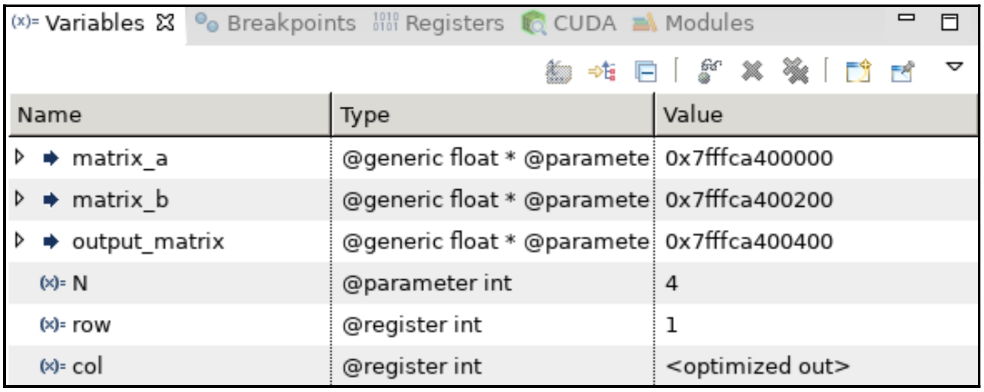

Locals
Name
Value
Type
@flatBlockIdx
0
long
@flatThreadIdx
1
long
threadIdx
{x = 1, y = 0, z = 0}
const uin
x
1
unsigned
y
0
unsigned
z
0
unsigned
blockIdx
{x = 0, y = 0, z = 0}
const uin
x
0
unsigned
y
0
unsigned
z
0
unsigned
blockDim
{x = 2, y = 2, z = 1}
const din
gridDim
{x = 2, y = 2, z = 1}
const din
@gridId
1
const lon
row
1
int
col
'col' has no value at the target location.
Autos
Locals
Watch 1


CUDA C/C++ Project @homecomputer
New CUDA C/C++ Project
Create a new project of selected type
Project name: matrix_ker
Use default location
Location: /home/mp/cuda-workspace/matrix_ker
Browse...
Choose file system: default
Project type:
Toolchains:
Executable
Empty Project
CUDA Runtime Project
Import CUDA Sample
Thrust Project
Shared Library
Static Library
Makefile project
CUDA Toolkit 9.2
Show project types and toolchains only if they are supported on the platform
< Back
Next >
Cancel
Finish


Problems
Tasks
Console
Properties
<terminated> matrix_ker [C/C++ Application] /home/mp/cuda-workspace/matrix_ker/Debug/matrix_ker (8/19/18 5:07 PM)
Success! Output of kernel matches expected output.

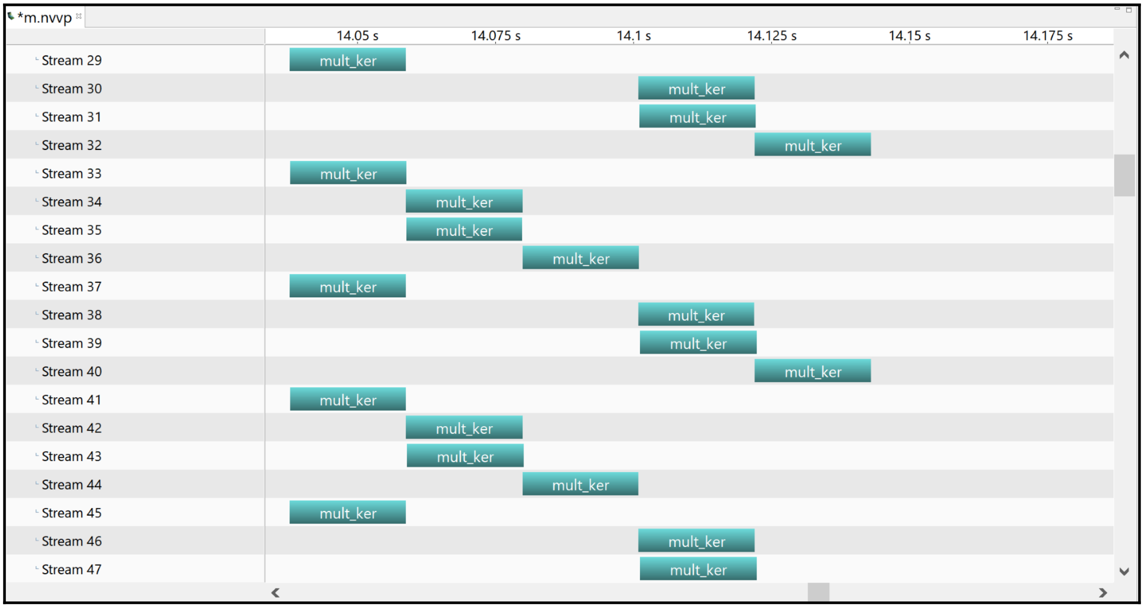

Variables
Breakpoints
Registers
CUDA
Modules
Search CUDA Information
[0] matrix_mult_ker | Device 0 (GP104-A) | 4 blocks of 4 are running
(0,0,0) | SM 0 | 4 threads of 4 are running
(0,0,0) | Warp 0 Lane 0 | matrix_ker.cu:43 (0x555555ca7910)
(0,1,0) | Warp 0 Lane 2 | matrix_ker.cu:43 (0x555555ca7910)
(1,0,0) | Warp 0 Lane 1 | matrix_ker.cu:43 (0x555555ca7910)
(1,1,0) | Warp 0 Lane 3 | matrix_ker.cu:43 (0x555555ca7910)
(0,1,0) | SM 2 | 4 threads of 4 are running
(1,0,0) | SM 1 | 4 threads of 4 are running
(1,1,0) | SM 3 | 4 threads of 4 are running


Variables
Breakpoints
Registers
CUDA
Modules
Name | Type | T(0,0,0)B(0,0,0)
matrix_a | @generic float * @parameter | 0x7fffca400000
matrix_b | @generic float * @parameter | 0x7fffca400200
output_matrix | @generic float * @parameter | 0x7fffca400400
N | @parameter int | 4
row | @register int | 0
col | @register int | <optimized out>


Variables
Breakpoints
Registers
CUDA
Modules
Name | Type | Value
matrix_a | @generic float * @paramete | 0x7fffca400000
matrix_b | @generic float * @paramete | 0x7fffca400200
output_matrix | @generic float * @paramete | 0x7fffca400400
N | @parameter int | 4
row | @register int | 1
col | @register int | <optimized out>

```
PS C:\Users\btuom\examples\6> .\divergence_test.exe
threadIdx.x 0 : This is an even thread.
threadIdx.x 2 : This is an even thread.
threadIdx.x 4 : This is an even thread.
threadIdx.x 6 : This is an even thread.
threadIdx.x 8 : This is an even thread.
threadIdx.x 10 : This is an even thread.
threadIdx.x 12 : This is an even thread.
threadIdx.x 14 : This is an even thread.
threadIdx.x 16 : This is an even thread.
threadIdx.x 18 : This is an even thread.
threadIdx.x 20 : This is an even thread.
threadIdx.x 22 : This is an even thread.
threadIdx.x 24 : This is an even thread.
threadIdx.x 26 : This is an even thread.
threadIdx.x 28 : This is an even thread.
threadIdx.x 30 : This is an even thread.
threadIdx.x 1 : This is an odd thread.
threadIdx.x 3 : This is an odd thread.
threadIdx.x 5 : This is an odd thread.
threadIdx.x 7 : This is an odd thread.
threadIdx.x 9 : This is an odd thread.
threadIdx.x 11 : This is an odd thread.
threadIdx.x 13 : This is an odd thread.
threadIdx.x 15 : This is an odd thread.
threadIdx.x 17 : This is an odd thread.
threadIdx.x 19 : This is an odd thread.
threadIdx.x 21 : This is an odd thread.
threadIdx.x 23 : This is an odd thread.
threadIdx.x 25 : This is an odd thread.
threadIdx.x 27 : This is an odd thread.
threadIdx.x 29 : This is an odd thread.
threadIdx.x 31 : This is an odd thread.
PS C:\Users\btuom\examples\6>
```

```
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   42.94%  2.3360us         2  1.1680us     896ns  1.4400us  [CUDA memcpy HtoD]
                   41.18%  2.2400us         1  2.2400us  2.2400us  2.2400us  matrix_mult_ker(float*, float*, float*, int)
                   15.88%     864ns         1     864ns     864ns     864ns  [CUDA memcpy DtoH]
      API calls:   72.42%  139.27ms         3  46.422ms  7.7580us  139.25ms  cudaMalloc
                   25.66%  49.351ms         1  49.351ms  49.351ms  49.351ms  cudaDeviceReset
                    1.46%  2.8053ms        88  31.878us     484ns  1.5375ms  cuDeviceGetAttribute
                    0.15%  290.91us         3  96.969us  14.060us  260.85us  cudaFree
                    0.14%  266.18us         3  88.727us  73.212us  111.52us  cudaMemcpy
                    0.06%  119.27us         1  119.27us  119.27us  119.27us  cuDeviceGetName
                    0.05%  101.33us         2  50.666us  11.152us  90.181us  cudaDeviceSynchronize
                    0.02%  38.787us         1  38.787us  38.787us  38.787us  cuDeviceTotalMem
                    0.02%  29.576us         1  29.576us  29.576us  29.576us  cudaLaunchKernel
                    0.01%  21.818us         1  21.818us  21.818us  21.818us  cudaSetDevice
                    0.00%  8.7280us         3  2.9090us     485ns  7.2730us  cuDeviceGetCount
                    0.00%  8.7270us         1  8.7270us  8.7270us  8.7270us  cuDeviceGetPCIBusId
                    0.00%  3.3940us         2  1.6970us     485ns  2.9090us  cuDeviceGet
```



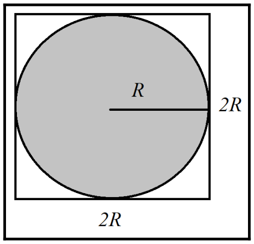
Chapter 7: Using the CUDA Libraries with Scikit-CUDA

```
In [3]: run cublas_gemm_flops.py
Single-precision performance: 1748.4264918 GFLOPS
Double-precision performance: 61.7956005349 GFLOPS
```

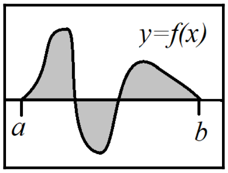

```
In [3]: print s**2
[  3.00100688e+05   1.00011516e+05   9.27482639e-03   9.26916022e-03
   9.21287015e-03   9.14533995e-03   9.02440213e-03   8.79677106e-03
   8.72804411e-03   8.61862674e-03]
```

```
In [4]: print u[:,0]
[ -7.07105637e-01   7.07107902e-01  -3.11021381e-06  -1.28881106e-06
  -2.04502885e-06   1.54779946e-06  -2.50539756e-06   3.35367849e-06
  -1.68121846e-06   3.36088988e-07]

In [5]: print u[:,1]
[ -7.07107902e-01  -7.07105637e-01  -5.94599987e-06   1.07432527e-07
   3.17310139e-07  -6.16845739e-07  -9.59202112e-07  -8.96491883e-07
   2.71546742e-06   6.28509406e-06]
```

Chapter 8: The CUDA Device Function Libraries and Thrust




```
In [25]: run monte_carlo_pi.py
Our Monte Carlo estimate of Pi is : 3.14159237769
NumPy's Pi constant is: 3.14159265359
Our estimate passes NumPy's 'allclose' : True
```



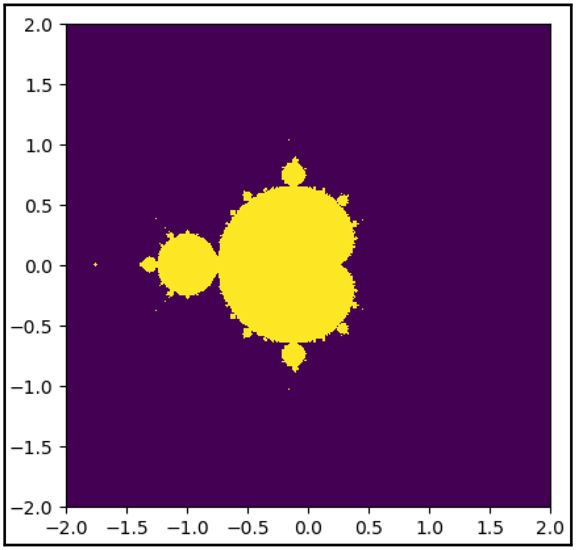

```
In [1]: code_string="%(precision)s x, y; %(precision)s * z;"

In [2]: code_dict = {'precision' : 'double'}

In [3]: code_double = code_string % code_dict

In [4]: print code_double
double x, y; double * z;
```

```
In [2]: sin_integral = MonteCarloIntegrator()

In [3]: sin_integral.definite_integral()
Out[3]: 2.0000000334270522
```

```
In [2]: run monte_carlo_integrator.py
The Monte Carlo numerical integration of the function
         f: x -> y =log(x)*_P2(sin(x))
         from x = 11.733 to x = 18.472 is : 8.9999892677
where the expected value is : 8.9999

The Monte Carlo numerical integration of the function
         f: x -> y = _R( 1 + sinh(2*x)*_P2(log(x)) )
         from x = 0.9 to x = 4 is : 0.584976671612
where the expected value is : 0.584977

The Monte Carlo numerical integration of the function
         f: x -> y = (cosh(x)*sin(x))/ sqrt( pow(x,3) + _P2(sin(x)))
         from x = 1.85 to x = 4.81 is : -3.34553137474
where the expected value is : -3.34553
```

```
PS C:\Users\btuom\examples\8> .\thrust_dot_product.exe
v[0] == 1
v[1] == 2
v[2] == 3
w[0] == 1
w[1] == 1
w[2] == 1
dot_product(v , w) == 6
```

Chapter 10: Working with Compiled GPU Code



Chapter 11: Performance Optimization in CUDA

```
PS C:\Users\btuom\examples\11> python .\dynamic_hello.py
Hello from thread 0, recursion depth 0!
Hello from thread 1, recursion depth 0!
Hello from thread 2, recursion depth 0!
Hello from thread 3, recursion depth 0!
Launching a new kernel from depth 0 .
-----------------------------------------
Hello from thread 0, recursion depth 1!
Hello from thread 1, recursion depth 1!
Hello from thread 2, recursion depth 1!
Launching a new kernel from depth 1 .
-----------------------------------------
Hello from thread 0, recursion depth 2!
Hello from thread 1, recursion depth 2!
Launching a new kernel from depth 2 .
-----------------------------------------
Hello from thread 0, recursion depth 3!
PS C:\Users\btuom\examples\11>
```

```
PS C:\Users\btuom\examples\11> python .\vectorized_memory.py
Vectorized Memory Test:
First int4: 1, 2, 3, 4
Second int4: 5, 6, 7, 8
First double2: 1.110000, 2.220000
Second double2: 3.330000, 4.440000
```

```
PS C:\Users\btuom\examples\11> python .\atomic.py
Atomic operations test:
add_out: 64
max_out: 127
```

```
PS C:\Users\btuom\examples\11> python .\shfl_xor.py
input array: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31]
array after __shfl_xor: [ 1  0  3  2  5  4  7  6  9  8 11 10 13 12 15 14 17 16 19 18 21 20 23 22 25
 24 27 26 29 28 31 30]
```

```
PS C:\Users\btuom\examples\11> python .\shfl_sum.py
Input array: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30 31]
Summed value: 496
Does this match with Pythons sum? : True
```

```
PS C:\Users\btuom\examples\11> python .\ptx_assembly.py
x is 123
x is now 0
x is now 1
f is now 3.330000
lane ID: 0
Do split64 / combine64 work? : true
```

```
In [14]: run sum_ker.py
Does sum_ker produces the same value as NumPy's sum (according allclose)? : True
Performing sum_ker / PyCUDA sum timing tests (20 each)...
sum_ker average time duration: 0.00553162831763, PyCUDA's gpuarray.sum average time duration: 0.0278831109579
(Performance improvement of sum_ker over gpuarray.sum: 5.04066964677 )
```