COMPUTE UNIFIED DEVICE ARCHITECTURE (CUDA)

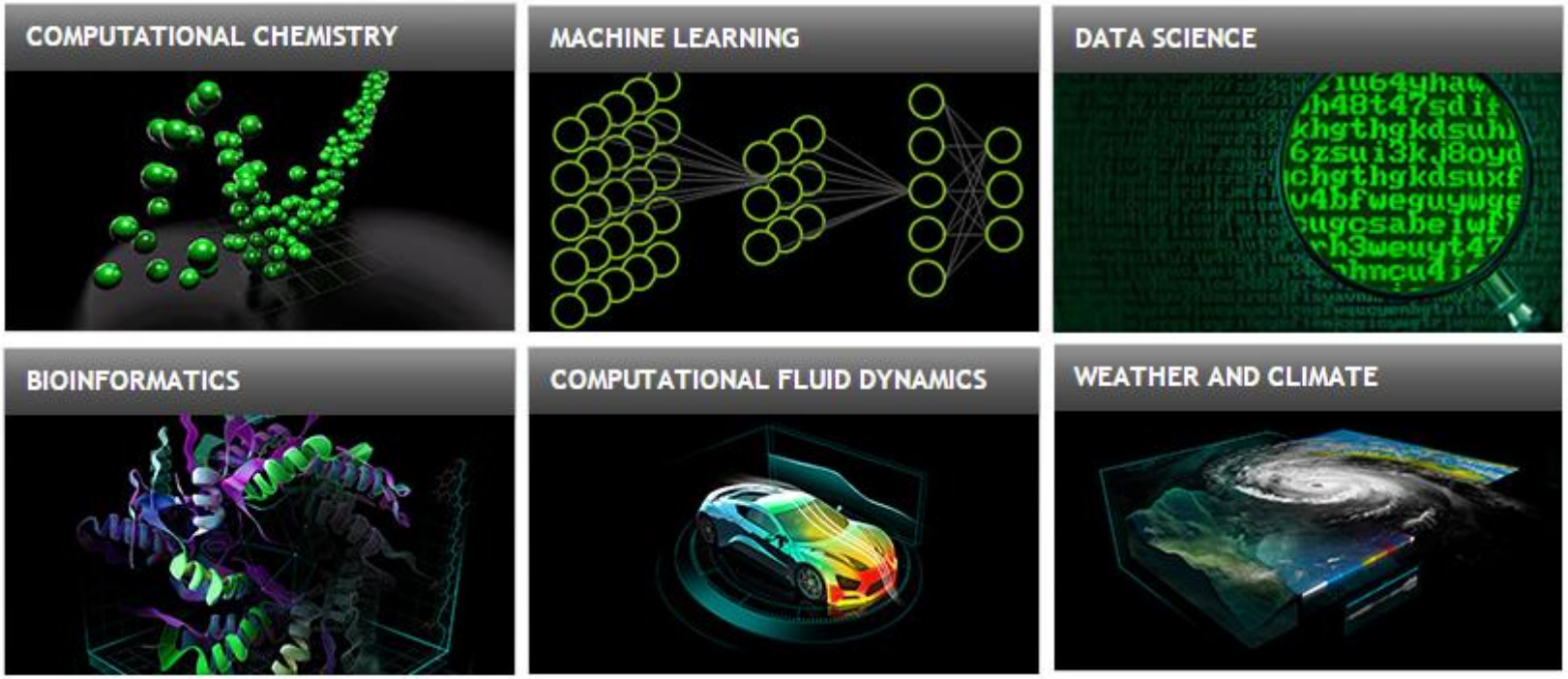
Francisco J. Hernández López

fcoj23@cimat.mx

TARJETA DE VIDEO O UNIDAD DE PROCESAMIENTO GRÁFICO (GPU)

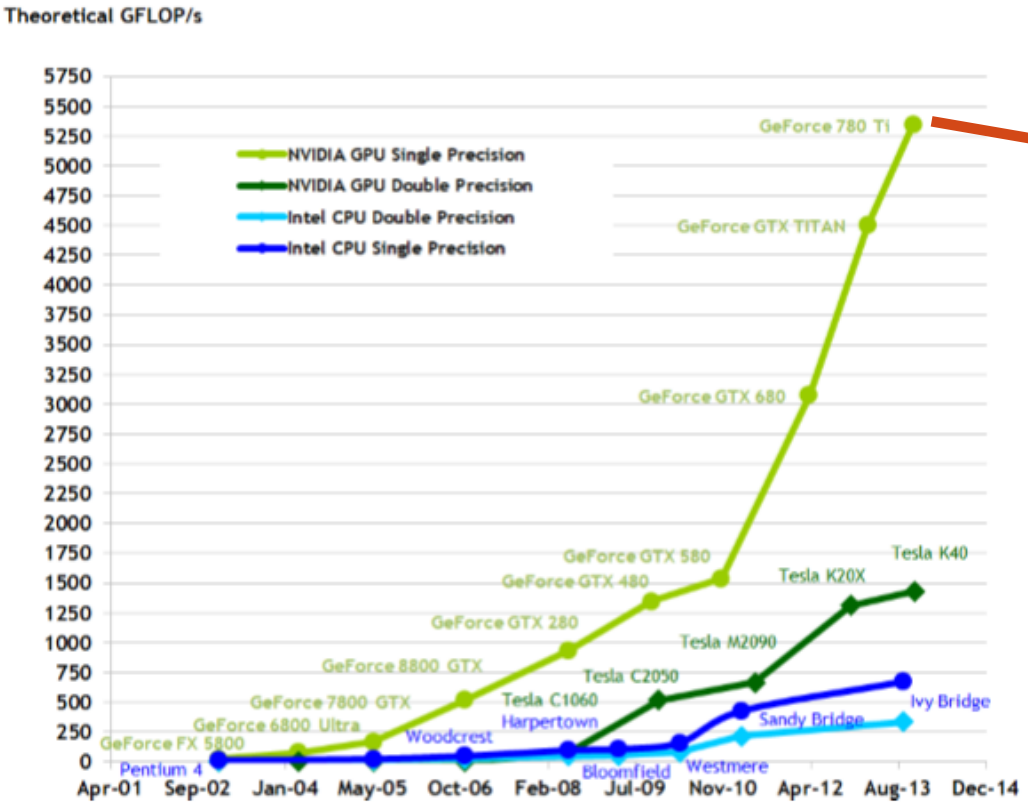
GPUS

- Procesadores flexibles de procesamiento general
- Se pueden resolver problemas de diversas áreas:
  - Finanzas, Gráficos, Procesamiento de Imágenes y Video, Algebra Lineal, Física, Química, Biología, etc.



Visitar CUDA ZONE: https://developer.nvidia.com/cuda-zone

GPUS VS CPUS

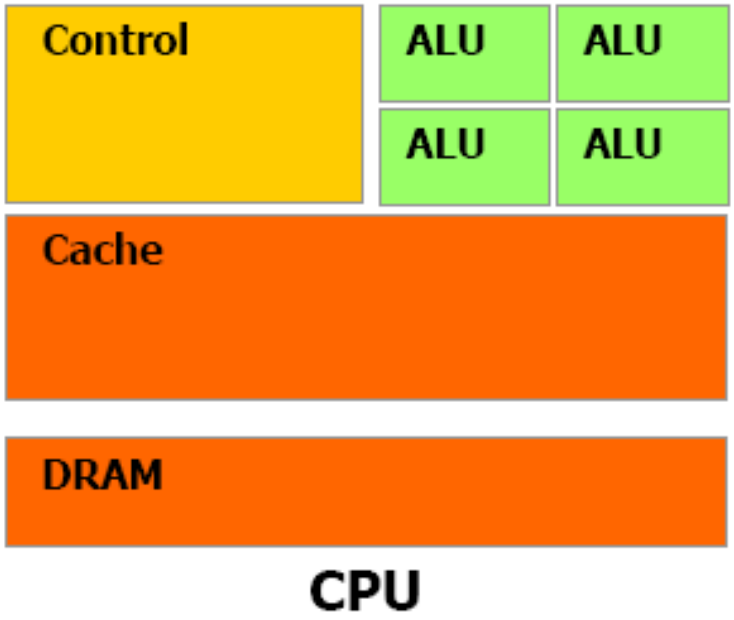


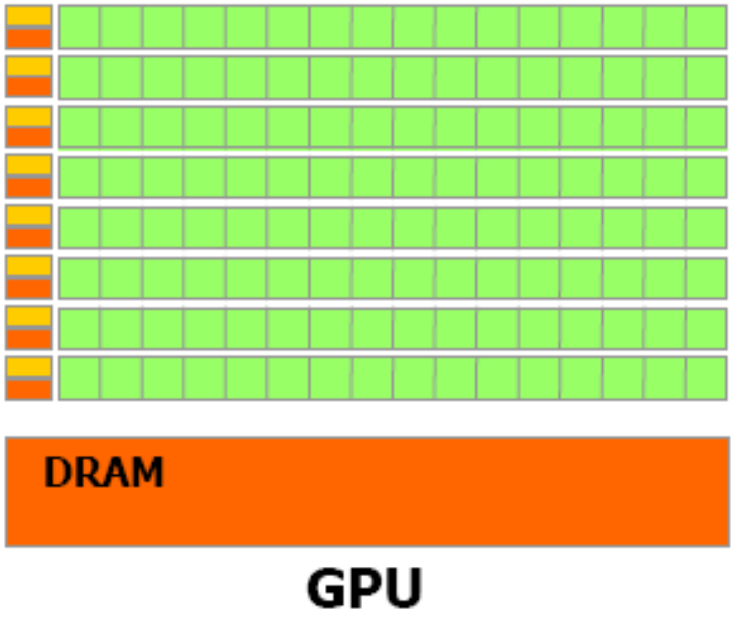
CUDA_C_Programming_Guide.pdf

GeForce GTX 780Ti

GPUs vs CPUs (C1)

- Las GPUs cuentan con mayor número de transistores para procesar los datos





CUDA_C_Programming_Guide.pdf

HERRAMIENTAS PARA CÓMPUTO PARALELO USANDO LA GPU

- **CUDA** (**C**ompute **U**nified **D**evice **A**rchitecture). Desarrollado por NVIDIA en el 2006, como uno de los primeros sistemas de programación en GPU que se liberaron de la forma que había para programar en una GPU (code-it-as-graphics, Cg). Compatible con GPUs Nvidia.
- **OpenCL** (**Open C**omputing **L**anguage). Es un estándar abierto para desarrollar programas que pueden ejecutarse en plataformas heterogéneas, incluyendo GPUs (Nvidia o AMD), CPU, DSPs (Digital Signal Processors). Su modelo de programación es muy parecido al de CUDA.
- **OpenACC**. Permite el uso de directivas para el compilador, para mapear automáticamente cálculos a la GPU o a un multicore.

Barlas, G. (2014). *Multicore and GPU Programming: An integrated approach*. Elsevier.

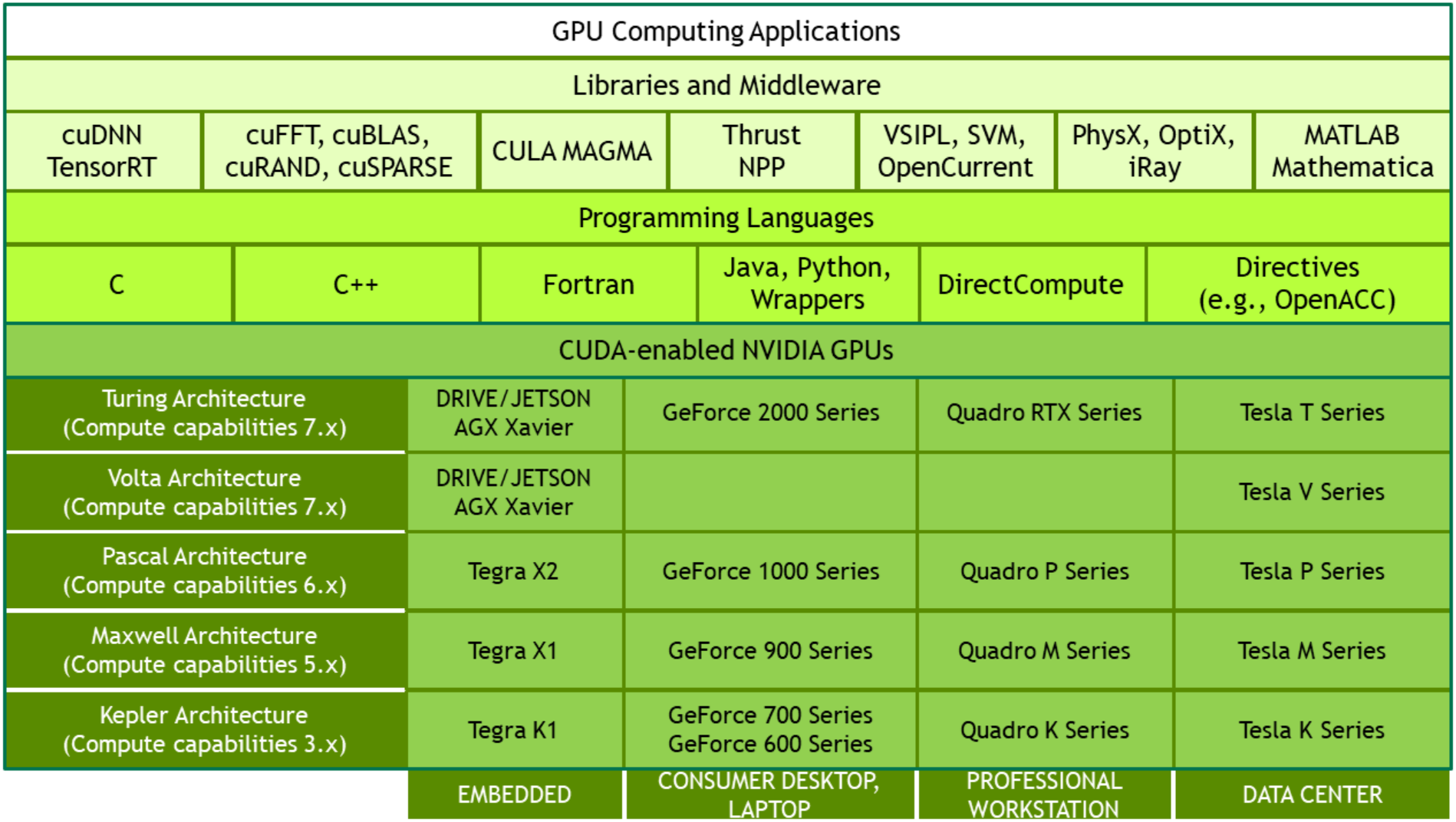
HERRAMIENTAS PARA CÓMPUTO PARALELO USANDO LA GPU (C1)

- **Thrust.** Es una librería de plantillas en C++ que acelera el desarrollo de programas en GPU, utilizando un conjunto de clases y un conjunto de algoritmos que automáticamente se ejecutan en la GPU. Desde la versión 1.6, puede lanzar ejecuciones a la GPU o a la CPU. Está incorporado en el SDK de CUDA desde la versión 4 de CUDA.
- **ArrayFire**. Es una librería completa de funciones para el GPU que cubre: Matemáticas, Procesamiento de imágenes y señales, Estadística, y otros dominios científicos. Opera en arreglo de datos de forma similar que Thrust.
- **C++ AMP** (C++ **A**ccelerated **M**assive **P**arallelism). Tecnología de Microsoft basado en DirectX 11. Permite la ejecución transparente del código C++ en una CPU o una GPU con base en un conjunto de directivas o extensiones del lenguaje. El modelo de programación es similar al de OpenMP.

Barlas, G. (2014). *Multicore and GPU Programming: An integrated approach*. Elsevier.

CUDA

- Es una tecnología de propósito general que nos permite ejecutar código en GPUs para hacer Cómputo Paralelo
- Desarrollado por NVIDIA en el 2006
- Soporta los lenguajes de programación C/C++, Fortran, Matlab, Python, LabView, etc.
- Soporte de datos en paralelo y manejador de hilos.
- Librerías:
  - FFT (Fast Fourier Transform)
  - BLAS (Basic Linear Algebra Subroutines)
  - CURAND (Generar números aleatorios)
  - CUSPARSE (Subrutinas de algebra lineal para operar matrices ralas)
  - NPP (NVIDIA Performance Primitives)…
- Opera internamente con OpenGL y DirectX.
- Soporta los sistemas operativos:
  - Windows XP 32/64-bit, Windows Vista 32/64-bit, Windows 7 32/64-bit, Linux 32/64-bit y Mac OS.

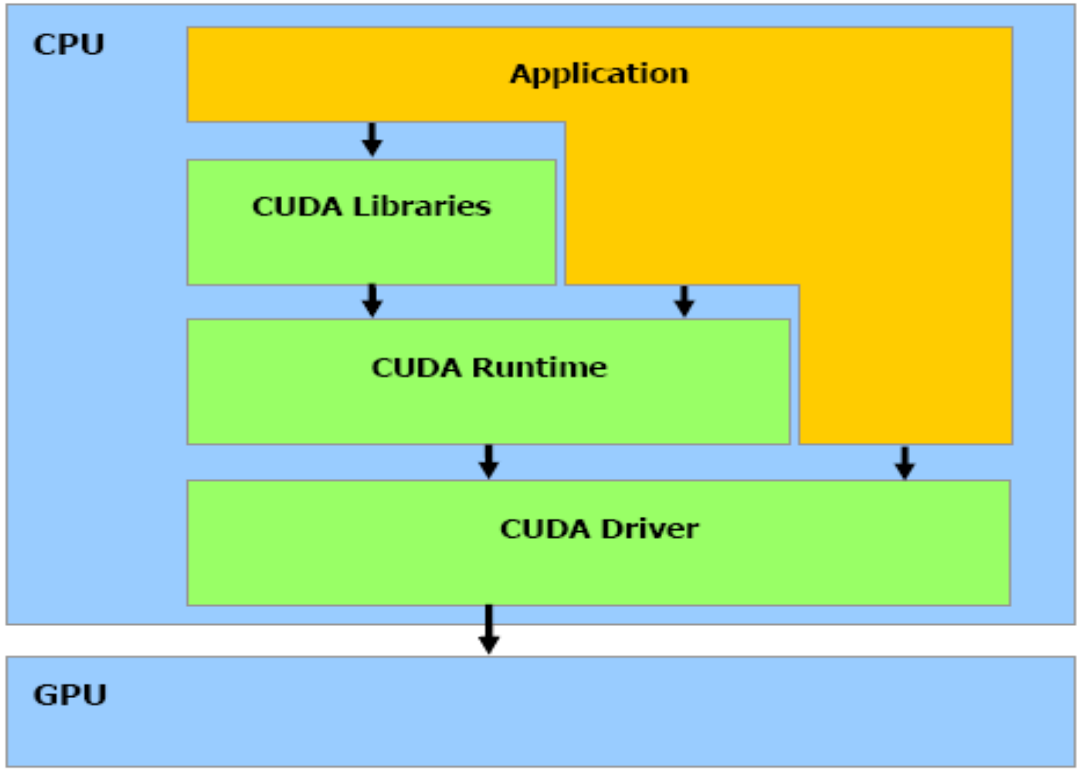


CUDA_C_Programming_Guide.pdf

SOFTWARE USANDO CUDA

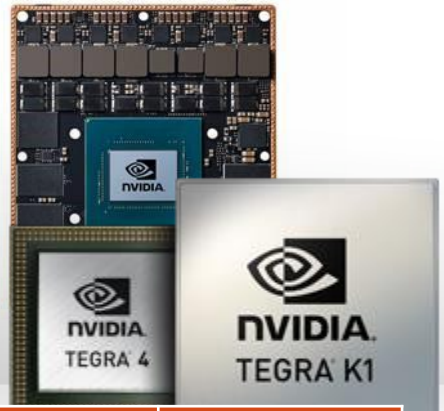
SOFTWARE CUDA

- El software CUDA esta compuesto por:
  - Hardware driver
  - Runtime
  - Libraries



CUDA_C_Programming_Guide.pdf

GPUS COMPATIBLES CON CUDA



Arquitectura	Capacidad	Ejemplos	Año
Tesla	1.0 - 1.3	GeForce 8800 GT, Quadro FX 370	2006
Fermi	2.0 - 2.1	GeForce GTX 480, Quadro 2000	2010
Kepler	3.0 - 3.5	Tesla K20, NVS 510, Tegra K1	2012
Maxwell	5.0 - 5.2	GeForce GTX 980M, Quadro M6000	2014
Pascal	6.0 - 6.1	GeForce GTX 1080, Quadro P6000	2016
Volta	7.0	NVIDIA Titan V, Tesla V100	2017
Turing	7.5	Quadro RTX 8000, Tesla T4	2018
Ampere	8.0 - 8.6	GeForce RTX 3070-3090, NVIDIA A100	2020
Lovelace, Hopper, Blackwell	8.9	NVIDIA RTX 4090 (Video juegos) NVIDIA H100 (Centro de datos)	2022-2023

Compute Capability (CUDA SDK support vs. Microarchitecture)

CUDA SDK version(s)	Tesla	Fermi	Kepler (early)	Kepler (late)	Maxwell	Pascal	Volta	Turing	Ampere	Ada Lovelace	Hopper
1.0[32]	1.0 – 1.1										
1.1	1.0 – 1.1+x										
2.0	1.0 – 1.1+x										
2.1 - 2.3.1[33][34][35][36]	1.0 – 1.3										
3.0 - 3.1[37][38]	1.0 –	2.0									
3.2[39]	1.0 –	2.1									
4.0 - 4.2	1.0 –	2.1+x									
5.0 - 5.5	1.0 –			3.5							
6.0	1.0 –			3.5							
6.5	1.1 –				5.x						
7.0 - 7.5		2.0 –			5.x						
8.0		2.0 –				6.x					
9.0 - 9.2			3.0 –				7.0				
10.0 - 10.2			3.0 –					7.5			
11.0[40]				3.5 –					8.0		
11.1 - 11.4[41]				3.5 –					8.6		
11.5 - 11.7.1[42]				3.5 –					8.7		
11.8[43]				3.5 –							9.0
12.0 - 12.2					5.0 –						9.0

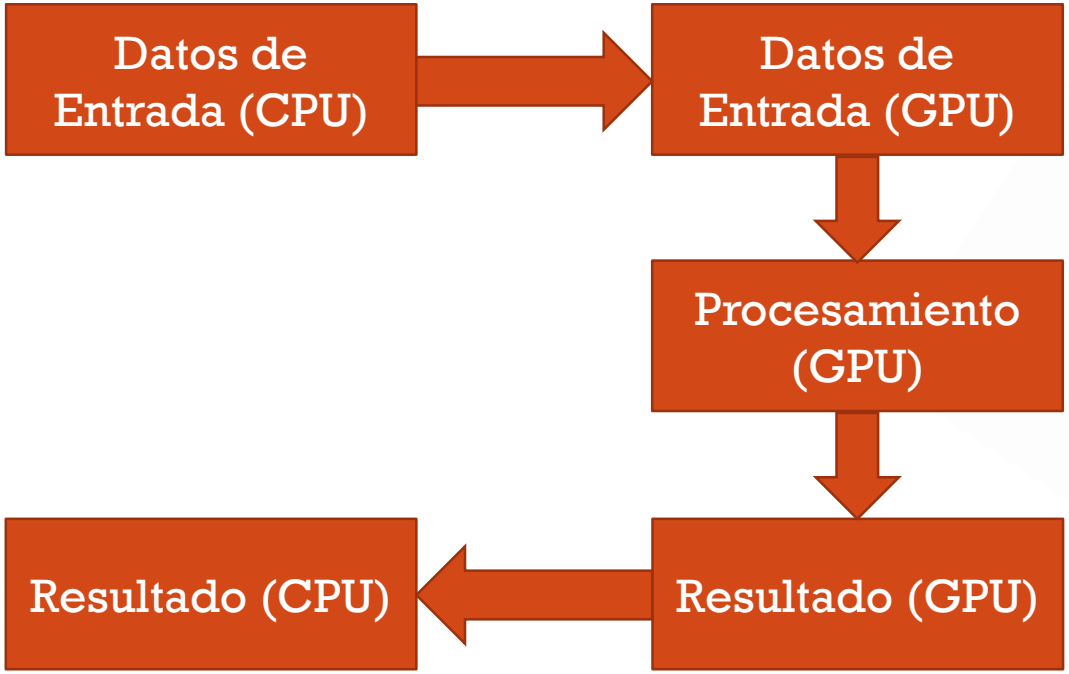
https://en.wikipedia.org/wiki/CUDA

Feature support (unlisted features are supported for all compute capabilities)	Compute capability (version)								
	1.0, 1.1	1.2, 1.3	2.x	3.0	3.2	3.5, 3.7, 5.x, 6.x, 7.0, 7.2	7.5	8.x	9.0
Warp vote functions (__all(), __any())	No	Yes							
Warp vote functions (__ballot())	No		Yes						
Memory fence functions (__threadfence_system())									
Synchronization functions (__syncthreads_count(), __syncthreads_and(), __syncthreads_or())									
Surface functions									
3D grid of thread blocks									
Warp shuffle functions	No		Yes						
Unified memory programming									
Funnel shift	No			Yes					
Dynamic parallelism	No				Yes				
Uniform Datapath [50]	No						Yes		
Hardware-accelerated async-copy	No							Yes	
Hardware-accelerated *split arrive/wait barrier*									
Warp-level support for reduction ops									
L2 cache residency management									
DPX instructions for accelerated dynamic programming	No								Yes
Distributed shared memory									
Thread block cluster									
Tensor memory accelerator (TMA) unit									
Feature support (unlisted features are supported for all compute capabilities)	1.0,1.1	1.2,1.3	2.x	3.0	3.2	3.5, 3.7, 5.x, 6.x, 7.0, 7.2	7.5	8.x	9.0
	Compute capability (version)								

https://en.wikipedia.org/wiki/CUDA

MODELO TRADICIONAL DE PROGRAMACIÓN EN CUDA

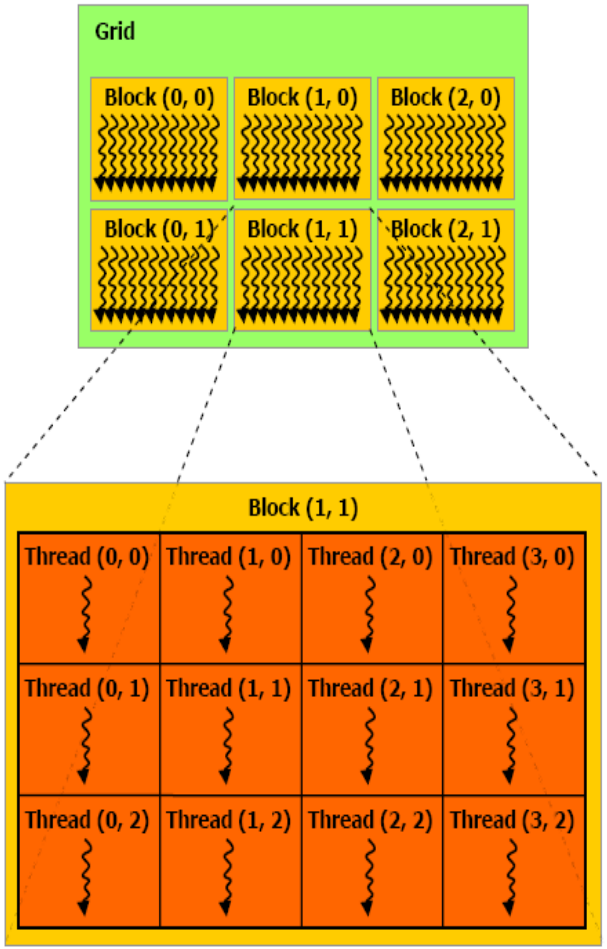
CPU



GPU

CONFIGURACIÓN DE LOS HILOS

- Un programa que se compila para ejecutarse en una tarjeta gráfica se le llama *Kernel*.
- El conjunto de hilos que ejecuta un *Kernel* están organizados como una cuadricula o malla (grid) de bloques de hilos.
- Un Bloque de hilos es un conjunto de hilos que pueden cooperar juntos:
  - Con rápido acceso a memoria compartida.
  - De forma sincronizada.
  - Con un identificador de hilos ID.
  - Los Bloques pueden ser arreglos de 1, 2 o 3 dimensiones.
- Un Grid de bloques de hilos:
  - Tiene un número limitado de hilos en un bloque.
  - Los bloques se identifican mediante un ID.
  - Pueden ser arreglos de 1 o 2 dimensiones. Hasta 3 en GPUs con Capacidad >=2



CUDA_C_Programming_Guide.pdf

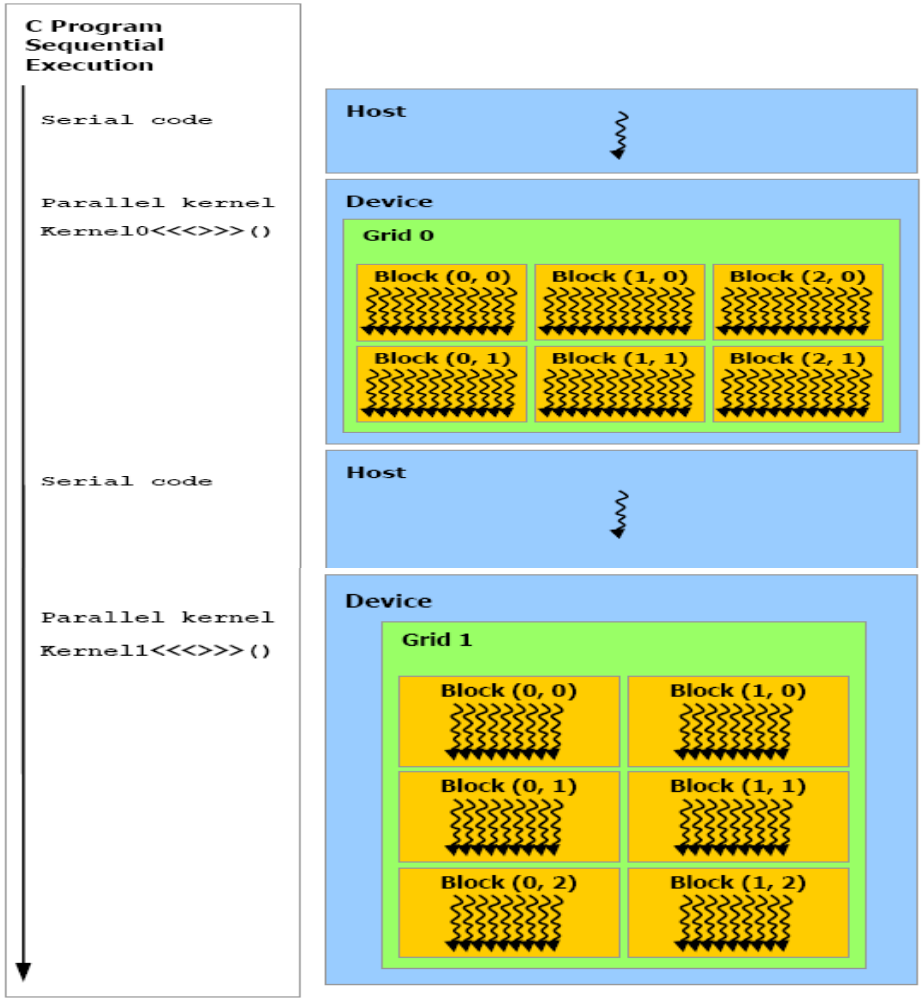
TAMAÑOS DE LOS BLOQUES Y MALLAS

Item	Compute Capability			
	1.x	2.x	3.x	5.x
Max. number of grid dimensions	2	3		
Grid maximum x-dimension	$2^{16} - 1$		$2^{31} - 1$	
Grid maximum y/z-dimension	$2^{16} - 1$			
Max. number of block dimensions	3			
Block max. x/y-dimension	512	1024		
Block max. z-dimension	64			
Max. threads per block	512	1024		
GPU example (GTX family chips)	8800	480	780	980

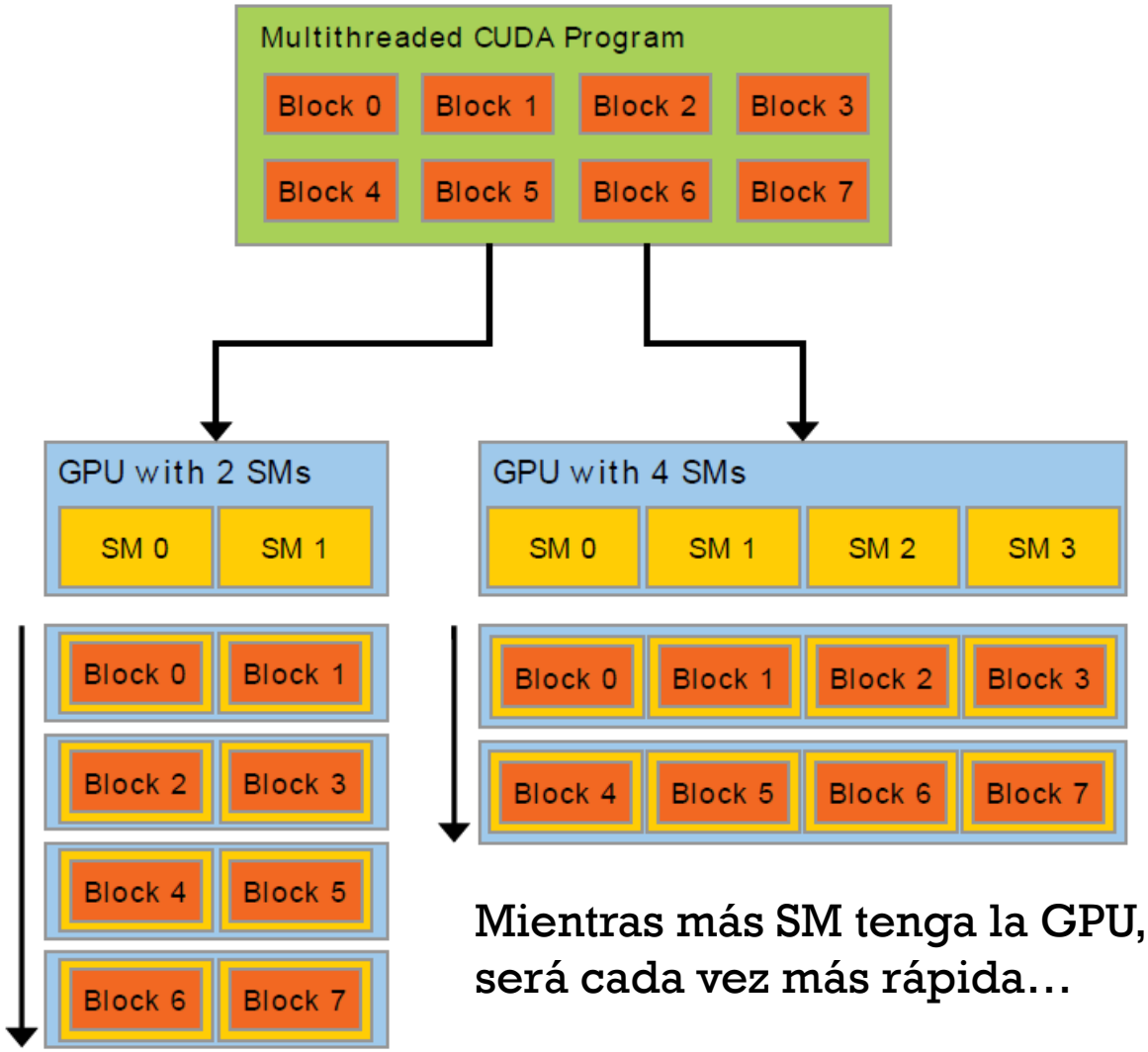
Barlas, G. (2014). *Multicore and GPU Programming: An integrated approach*. Elsevier.

EJECUCIÓN DE UN PROGRAMA EN GPU

Host = CPU
Device = GPU
Kernel = Conjunto de instrucciones que se ejecutan en el device.



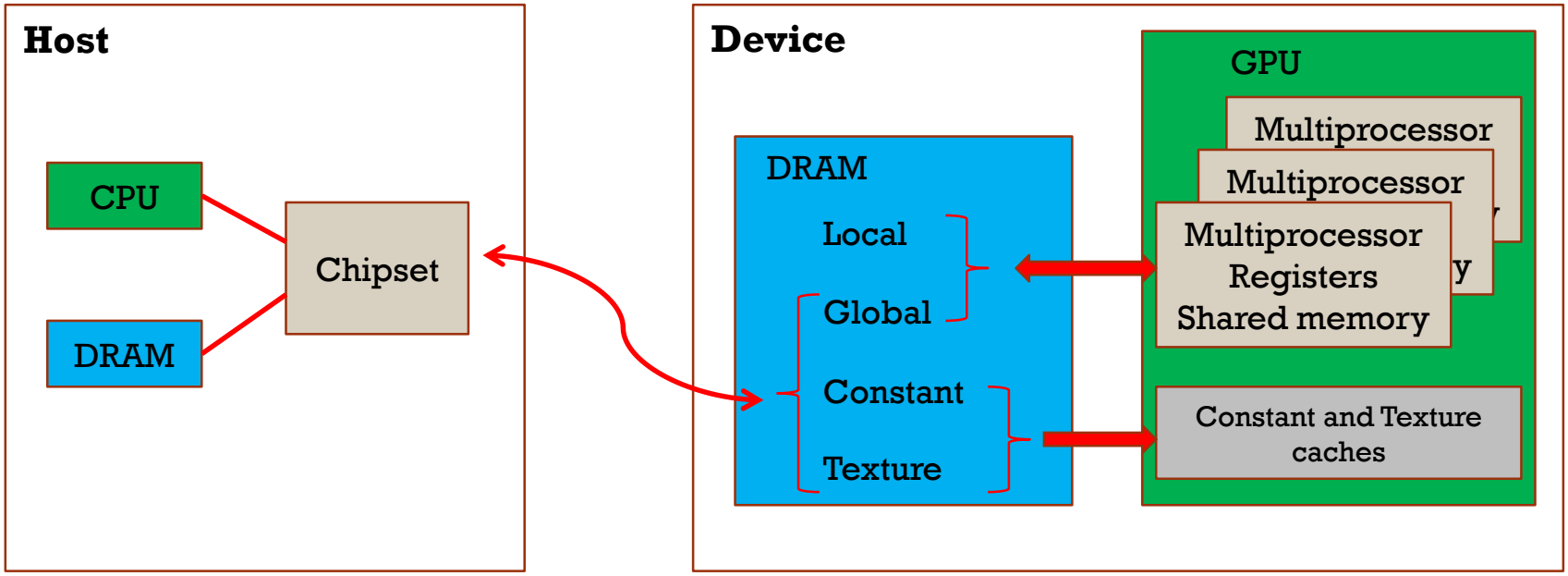
ESCALABILIDAD AUTOMÁTICA



Mientras más SM tenga la GPU, está será cada vez más rápida…

CUDA_C_Programming_Guide.pdf

MODELO DE LA MEMORIA EN CUDA



INSTRUCCIONES PARA CREAR MEMORIA

- **cudaMalloc** ((void**) devPtr, size_t size)
- **cudaMallocHost** ((void**) hostPtr, size_t size)
- **cudaFree** (void *devPtr)
- **cudaFreeHost** (void *hostPtr)

INSTRUCCIONES PARA COPIAR MEMORIA

- **cudaMemcpy**(void *dst, const void *src, size_t count, enum cudaMemcpyKind kind)
- **cudaMemcpy2D**(void *dst, size_t dpitch, const void *src, size_t spitch, size_t width, size_t height, enum cudaMemcpyKind kind)
- **cudaMemcpyToSymbol**(const char *symbol, const void *src,size_t count,size_t offset,enum cudaMemcpyKind kind) H→D D→D
- **cudaMemcpyFromSymbol**(void *dst, const char *symbol, size_t count, size_t offset, enum cudaMemcpyKind kind) D→H D→D

***Kind* =**
cudaMemcpyHostToHost = 0, cudaMemcpyHostToDevice = 1, cudaMemcpyDeviceToHost = 2, cudaMemcpyDeviceToDevice = 3. cudaMemcpyDefault = 4 (Unified Virtual Address)

CALIFICADORES DE UNA FUNCIÓN

__device__

- Se ejecuta en el dispositivo
- Llamada solamente desde el dispositivo

__global__

- Se ejecuta en el dispositivo
- Llamada solamente desde el host

__host__

- Se ejecuta en el host
- Llamada solamente desde el host

CALIFICADORES DE UNA VARIABLE

__device__

- Reside en el espacio de la memoria global
- Tiene el tiempo de vida de una aplicación
- Es accesible a partir de todos los hilos dentro del grid, y a partir del host a través de la biblioteca en tiempo de ejecución

__constant__ (Opcionalmente se utiliza junto con **__device__**)

- Reside en el espacio de la memoria constante
- Tiene el tiempo de vida de una aplicación
- Es accesible a partir de todos los hilos dentro del grid, y a partir del host a través de la biblioteca en tiempo de ejecución

__shared__ (Opcionalmente se utiliza junto con **__device__**)

- Reside en el espacio de memoria compartida de un bloque de hilos
- Tiene el tiempo de vida de un bloque
- Solamente accesible a partir de los hilos que están dentro del bloque

LLAMADA A UNA FUNCIÓN KERNEL

Una función, por ejemplo:

__global__ void NameFunc(float *parametro);

debe ser llamada como sigue:

NameFunc <<< Dg, Db, Ns, St >>> (parametro);

Dg: Es de tipo *dim3* dimensión y tamaño del grid
Db: Es de tipo *dim3* dimensión y tamaño de cada bloque
Ns: Es de tipo *size_t* número de bytes en memoria compartida
St: Es de tipo *cudaStream_t* el cuál indica que stream va a utilizar la función kernel

(Ns y St son argumentos opcionales)

VARIABLES DEFINIDAS AUTOMÁTICAMENTE

Todas las funciones __global__ y __device__ tienen acceso a las siguientes variables:

- **gridDim** es de tipo dim3, indica la dimensión del grid
- **blockIdx** es de tipo uint3, indica el índice del bloque dentro del grid
- **blockDim** es de tipo dim3, indica la dimensión del bloque
- **threadIdx** es de tipo uint3, indica el índice del hilo dentro del bloque

TIPOS DE DATOS

char1, uchar1, char2, uchar2, char3, char3, char4, uchar4, short1, ushort1, short2, ushort2, short3, ushort3, short4, ushort4, int1, uint1, int2, uint2, int3, uint3, int4, int4, long1, ulong1, long2, ulong2, long3, ulong3, long4, ulong4, longlong1, longlong2, float1, float2, float3, float4, double1,double2

La 1ra, 2da, 3ra, and 4ta componentes se acceden a través de los campos x, y, z y w respectivamente

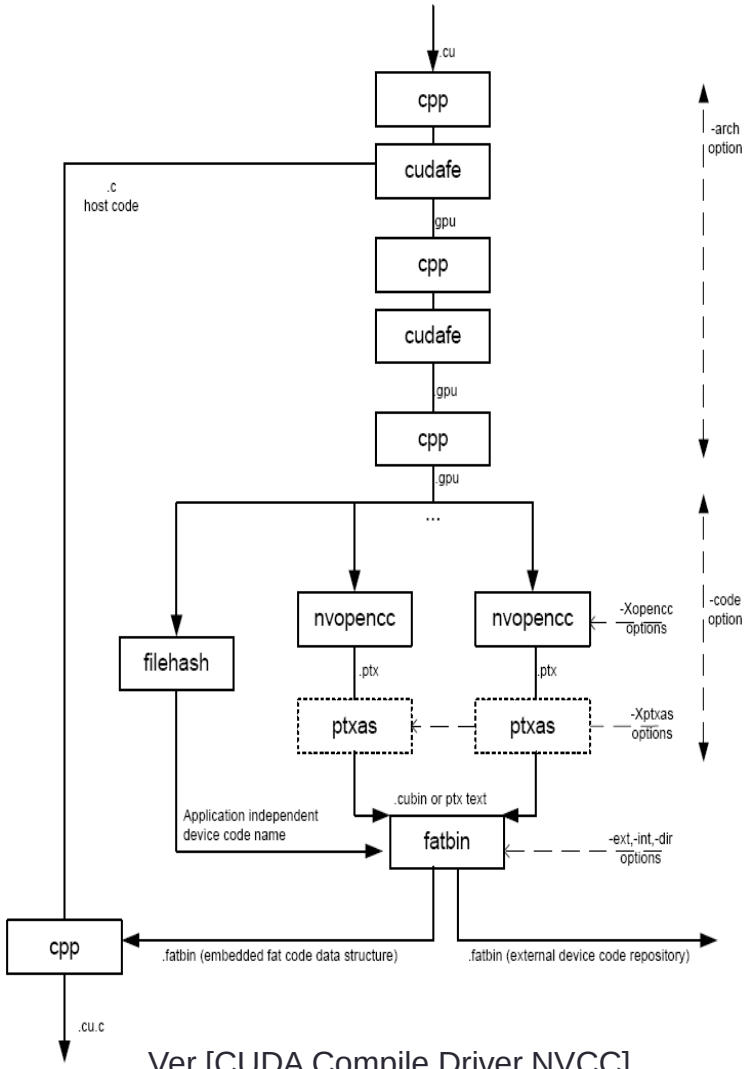
```
float3 temp[10];
……..
temp[i].x=0.0; temp[i].y=0.0; temp[i].z=0.0;
```

FUNCIONES MATEMÁTICAS

- __NombreFuncion()
  - A nivel de hardware
  - Mayor velocidad pero menor precisión
  - Ejemplos: __sinf(x), __expf(x), __logf(x),…
- NombreFuncion()
  - Menor velocidad pero mayor precisión
  - Ejemplos: sinf(x), expf(x), logf(x),…
- -use_fast_math: Opción del compilador nvcc

COMPILACIÓN CON NVCC

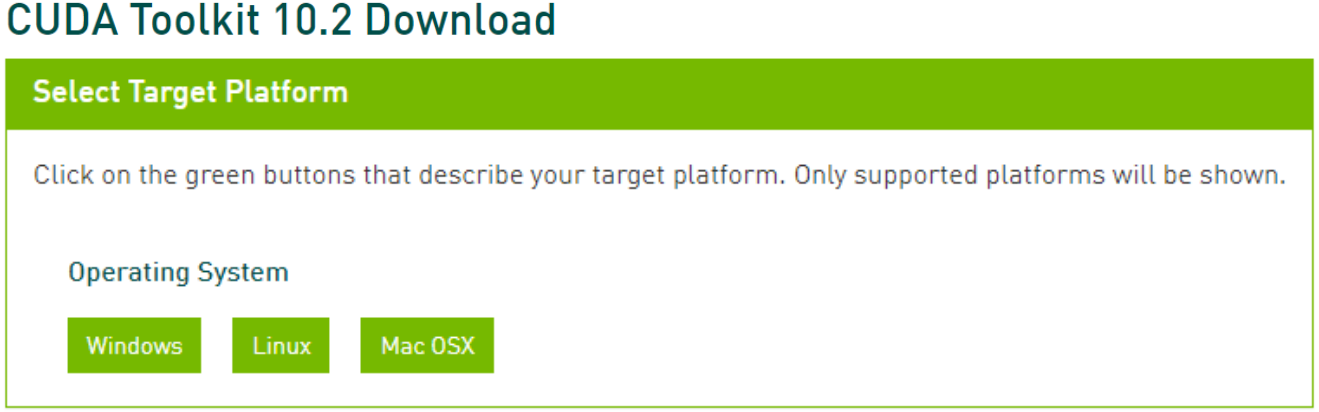
- El *nvcc*, es el encargado de compilar el código CUDA
- Soporta C/C++
- El *nvcc* utiliza los siguientes compiladores para el código *host*:
  - Linux: gcc, g++
  - Windows: Microsoft VS C/C++
  - Mac: Xcode

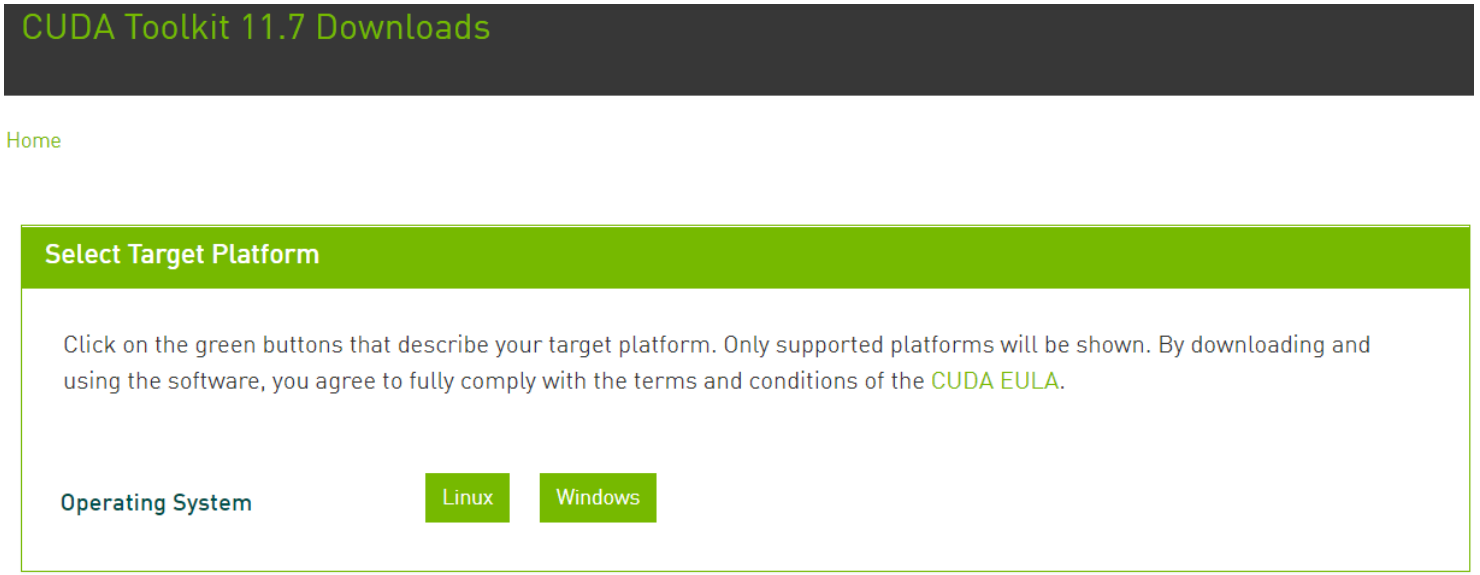


Ver [CUDA Compile Driver NVCC]

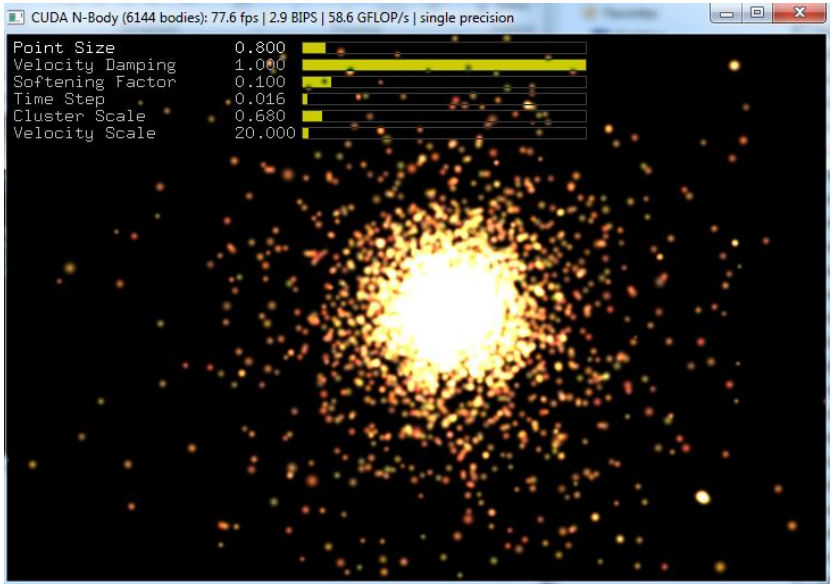
INSTALANDO CUDA

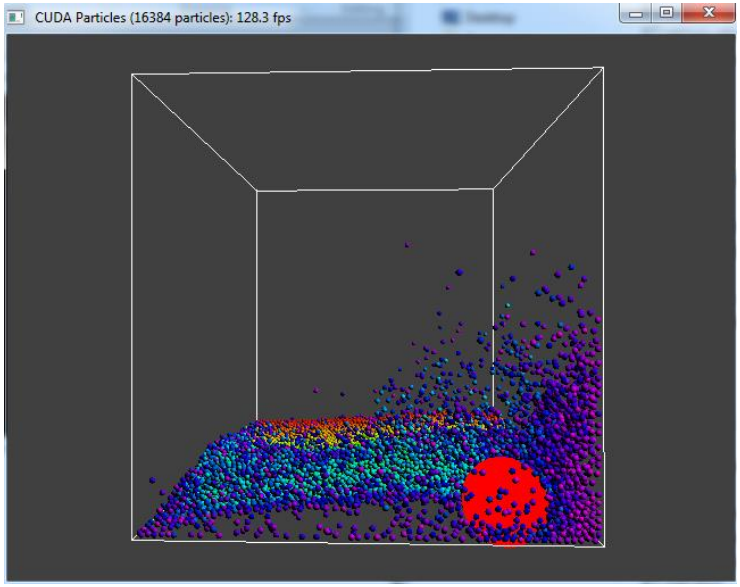
https://developer.nvidia.com/cuda-downloads

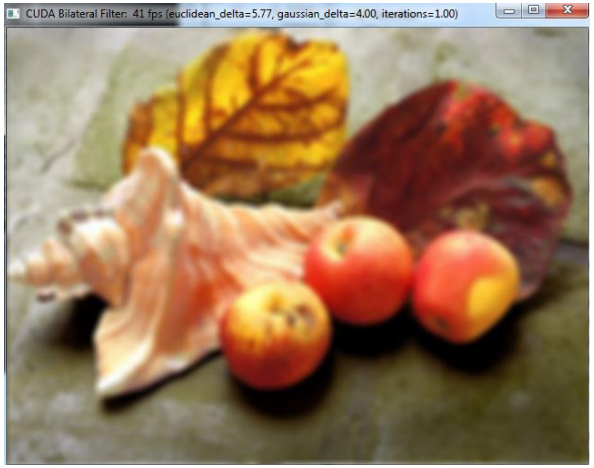


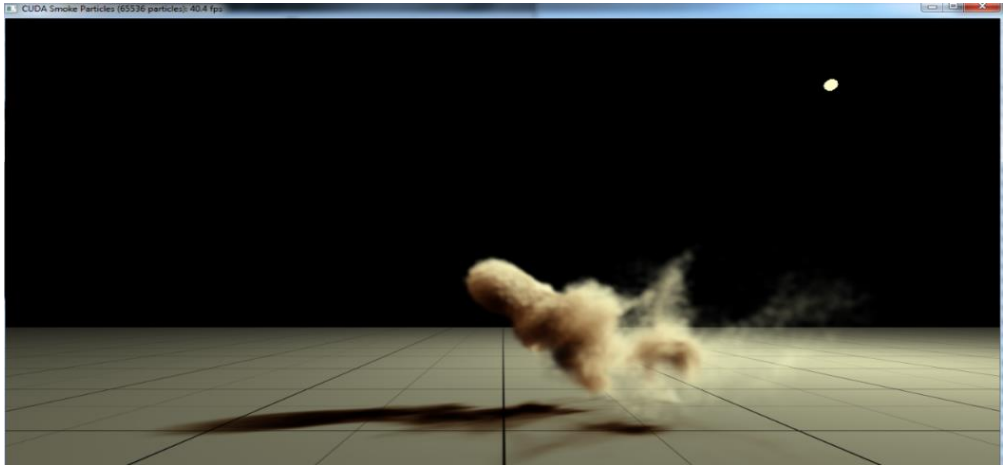


EJEMPLOS DEL SDK



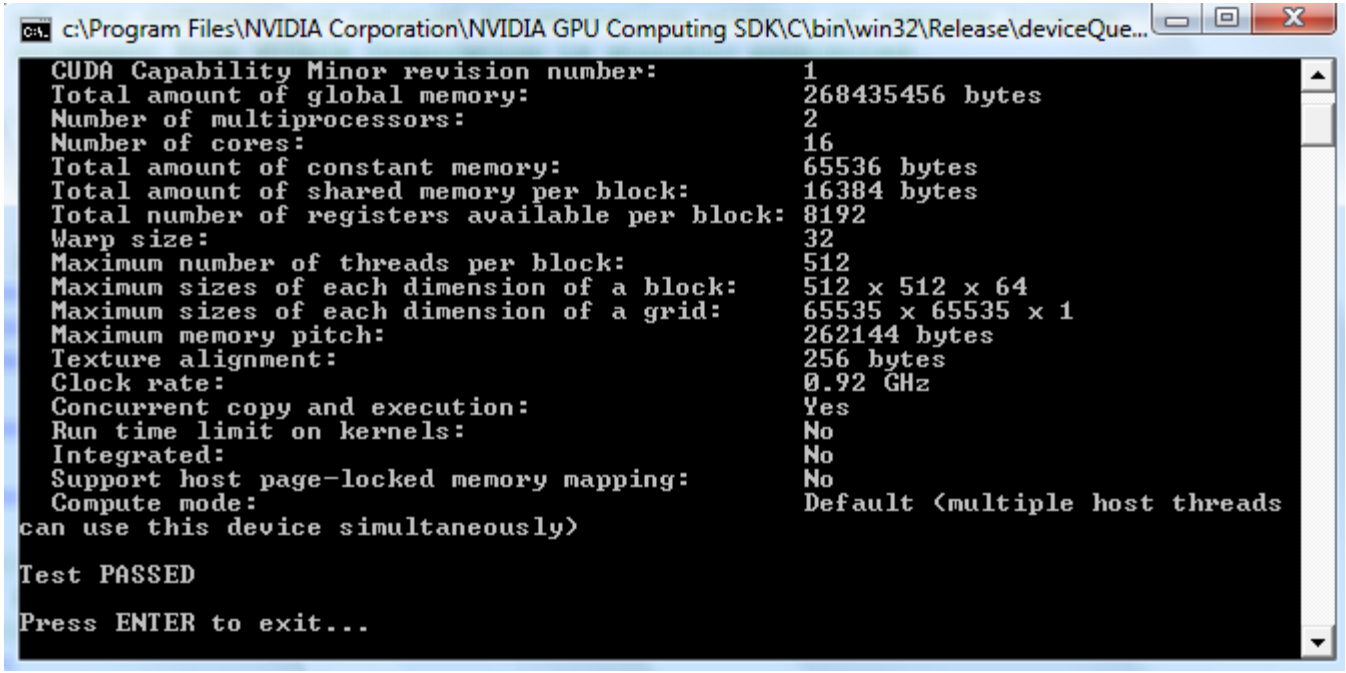






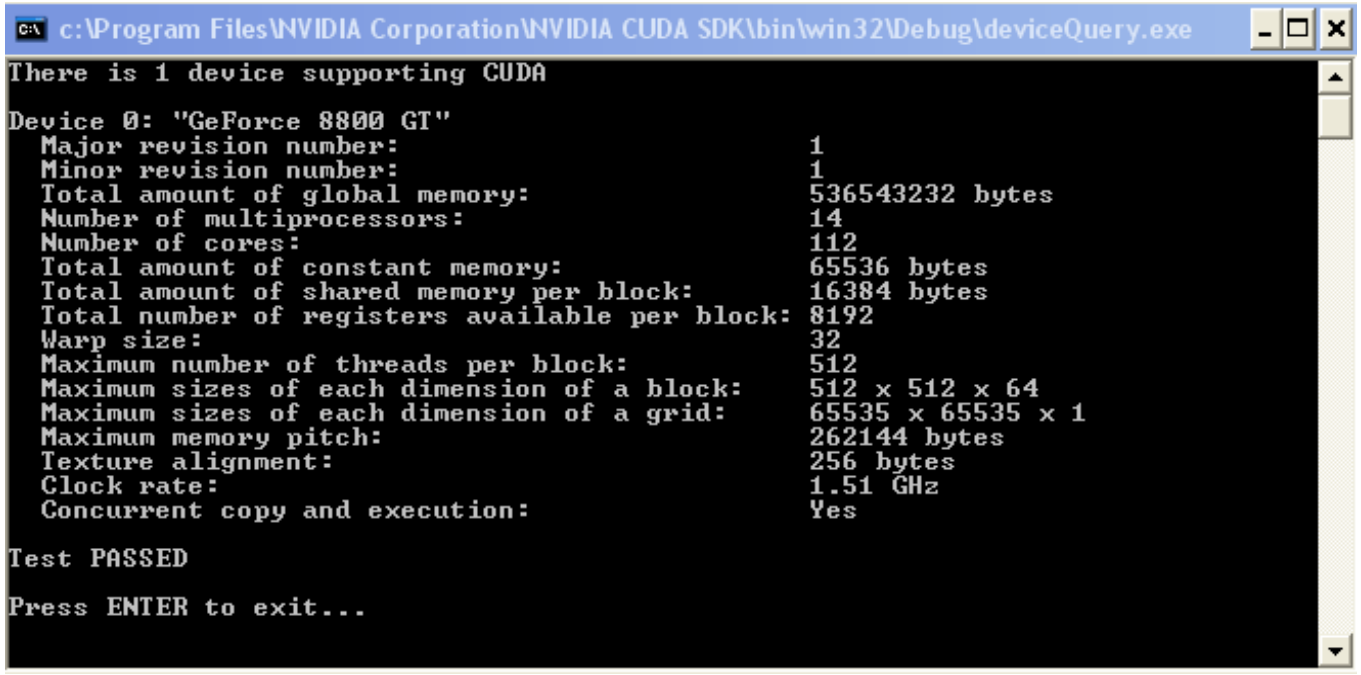
“deviceQueryDrv”

•Para saber que capacidades tiene nuestra tarjeta de video:



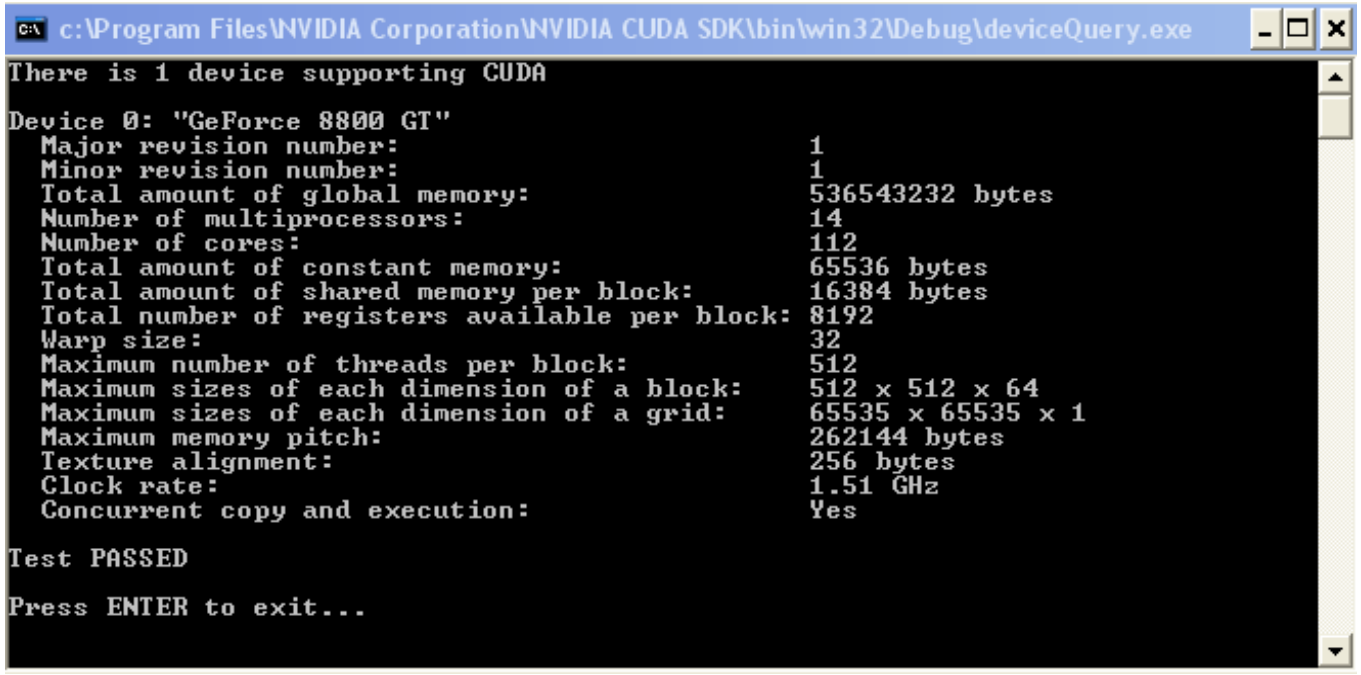
Resultado con una tarjeta NVIDIA GeForce 8400 GS

“deviceQueryDrv”



NVIDIA GeForce 8800 GT

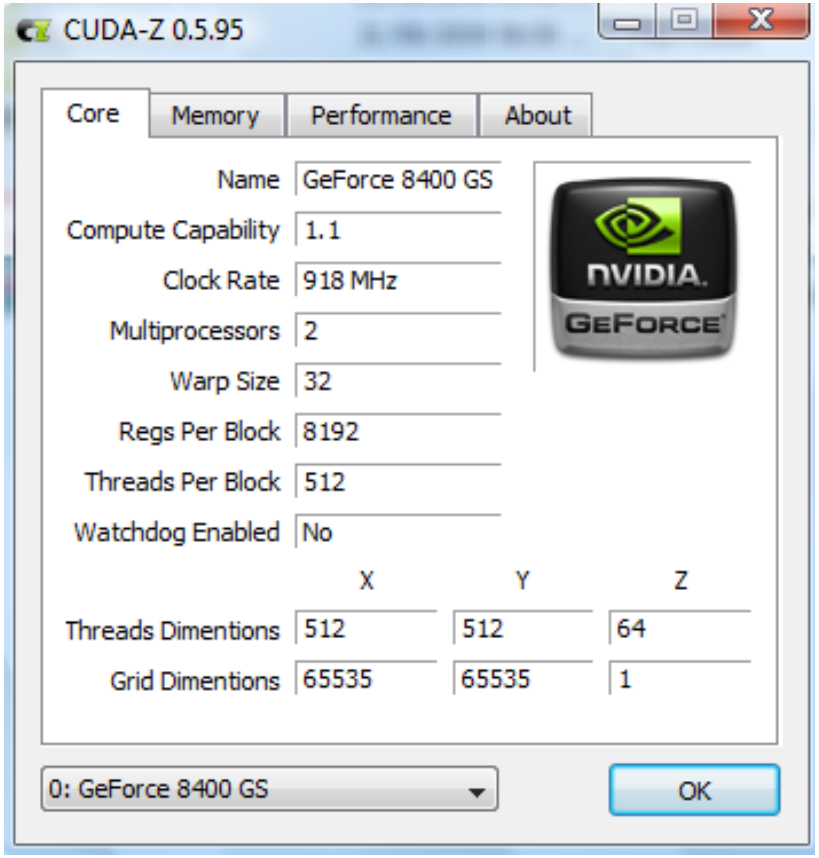
“deviceQueryDrv”

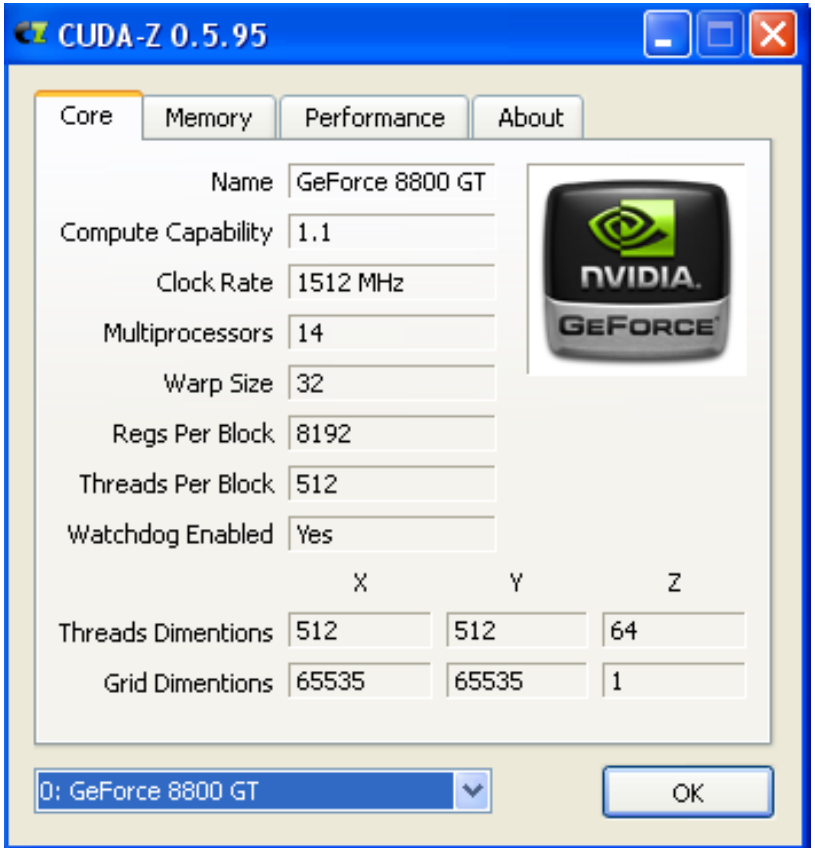


NVIDIA GeForce 8800 GT

CUDA-Z

- GeForce 8400 GS & GeForce 8800 GT

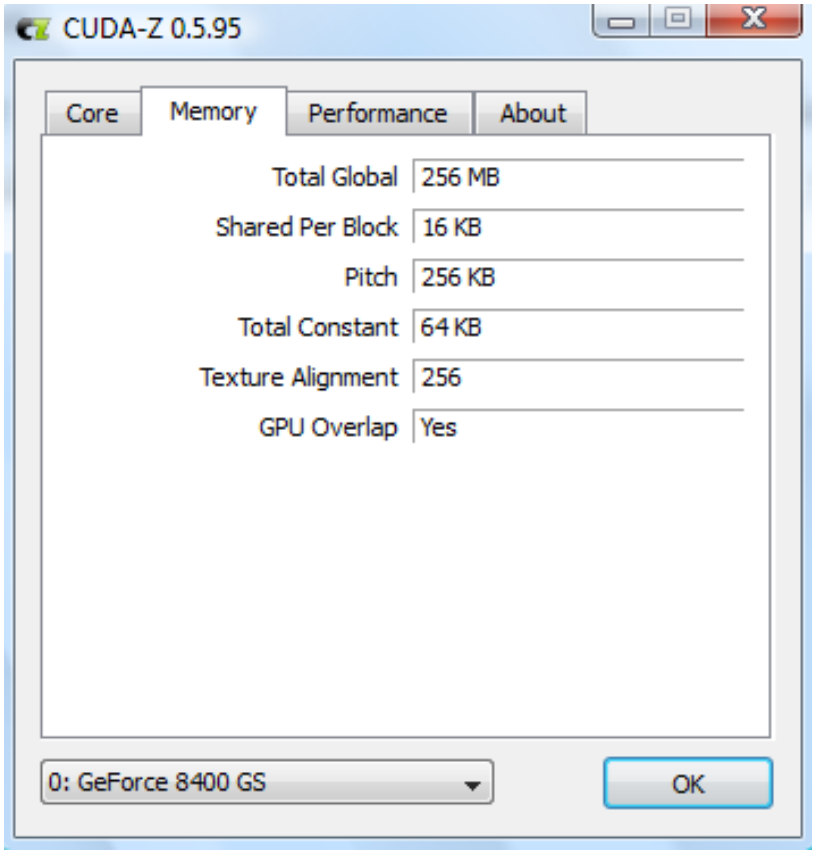


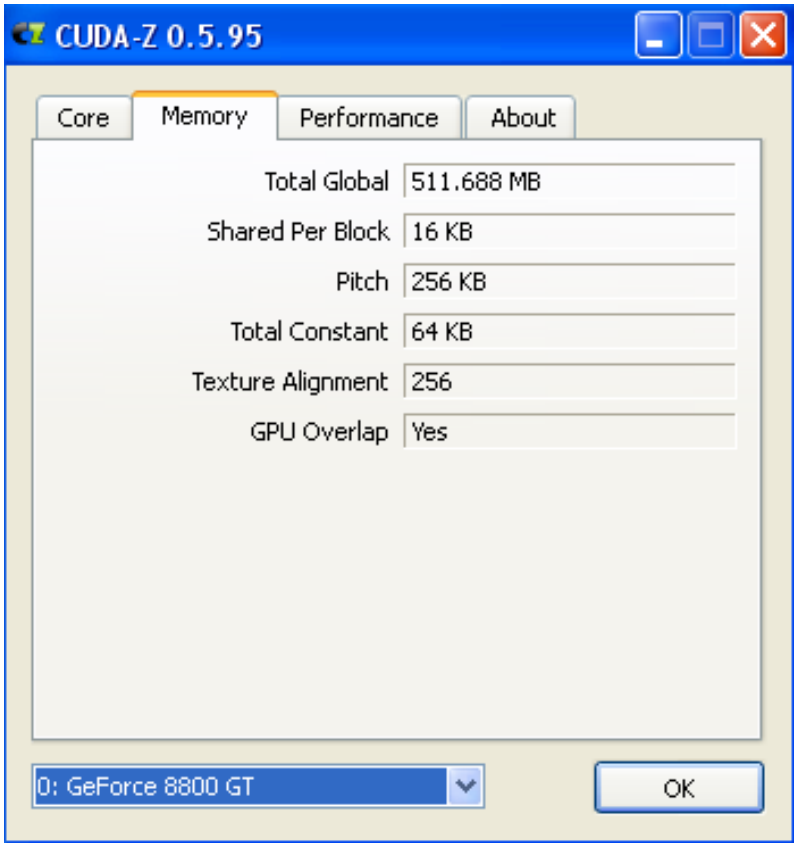


[http://cuda-z.sourceforge.net/]

CUDA-Z

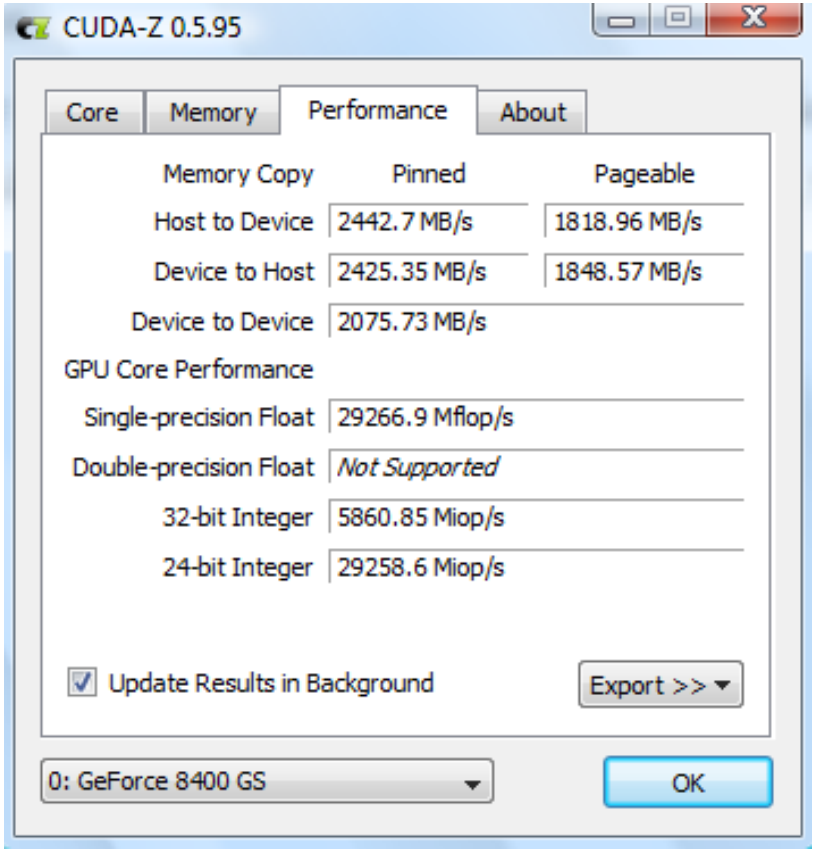
- GeForce 8400 GS & GeForce 8800 GT

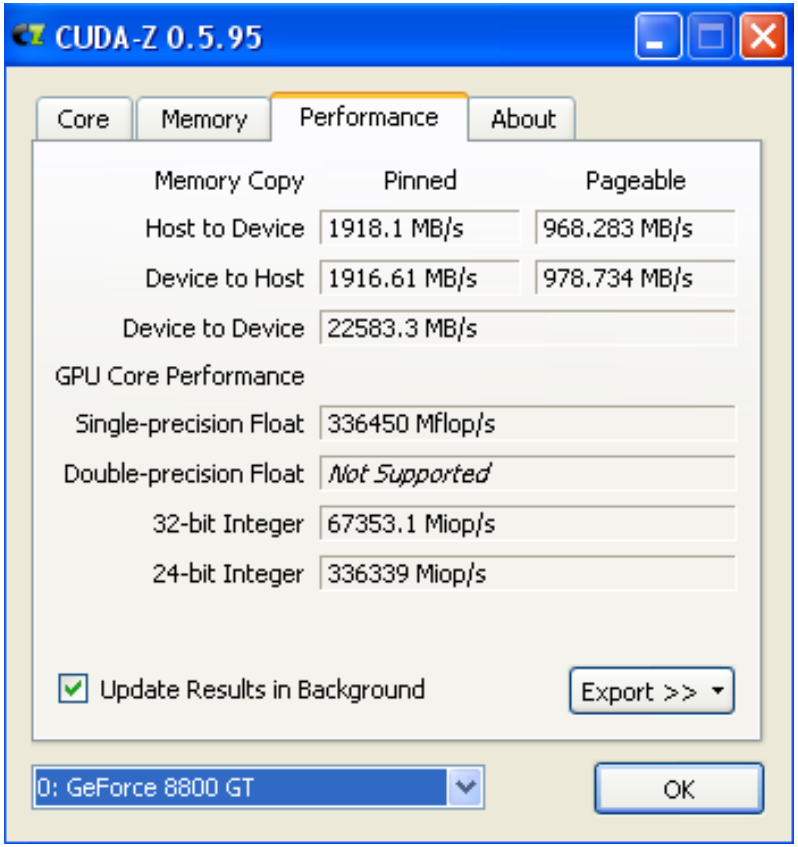




CUDA-Z

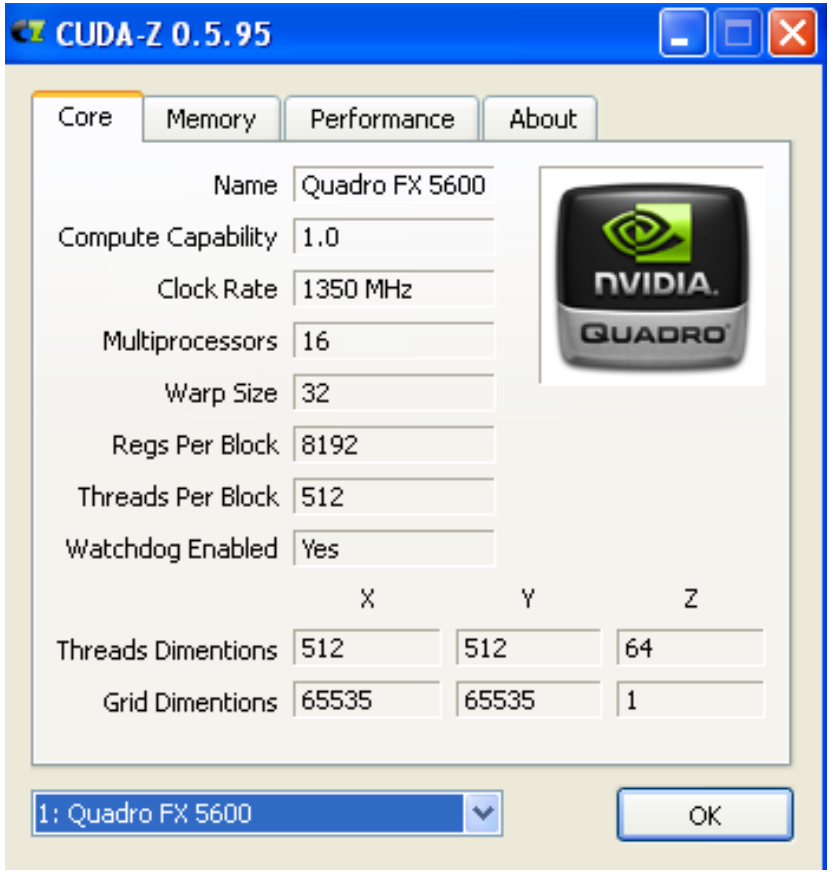
- GeForce 8400 GS & GeForce 8800 GT

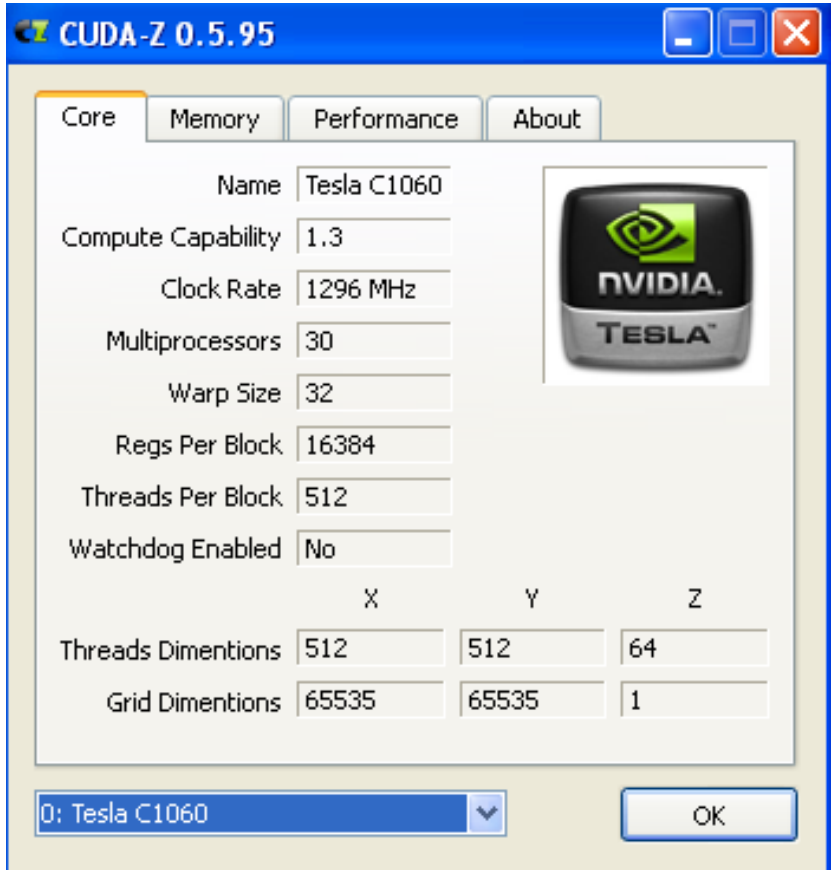




CUDA-Z

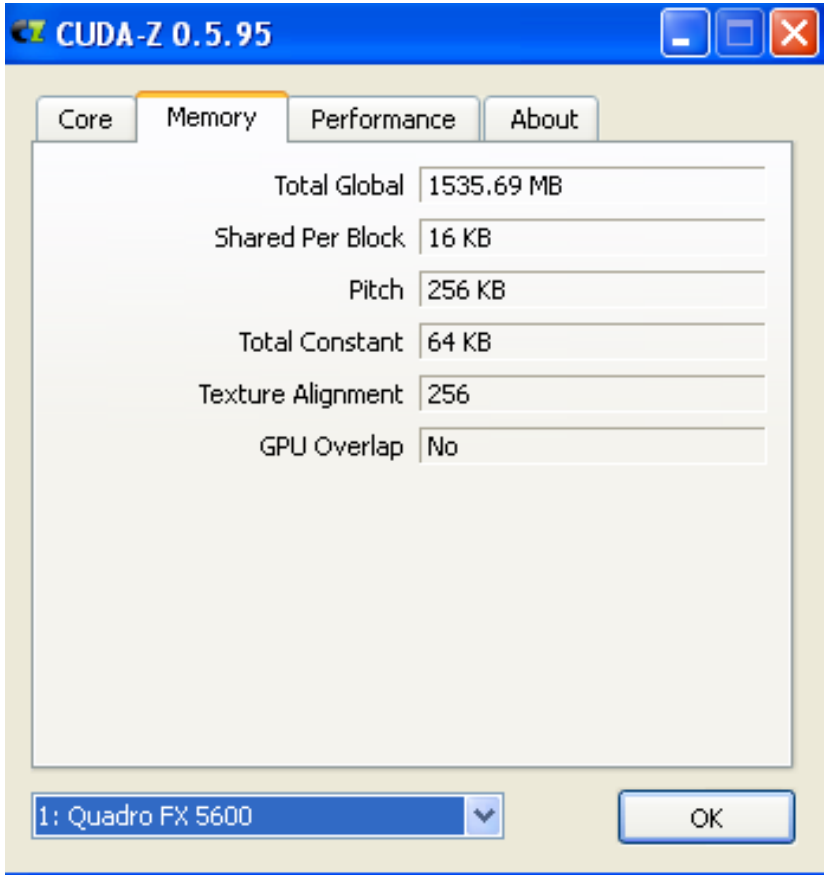
- Quadro FX 5600 & Tesla C1060

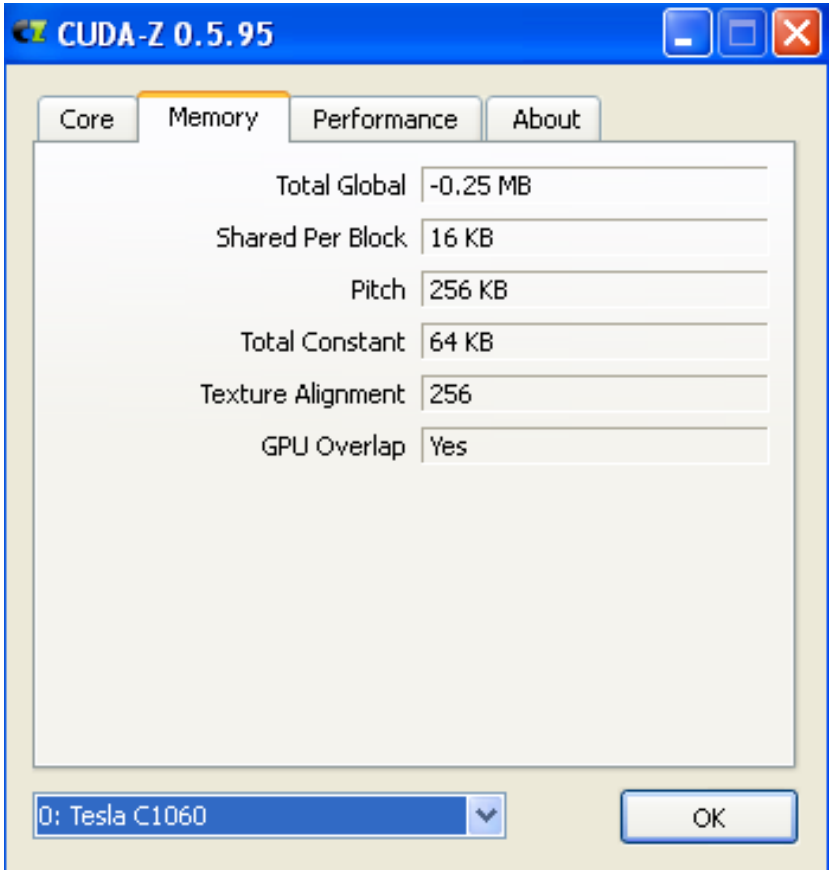




CUDA-Z

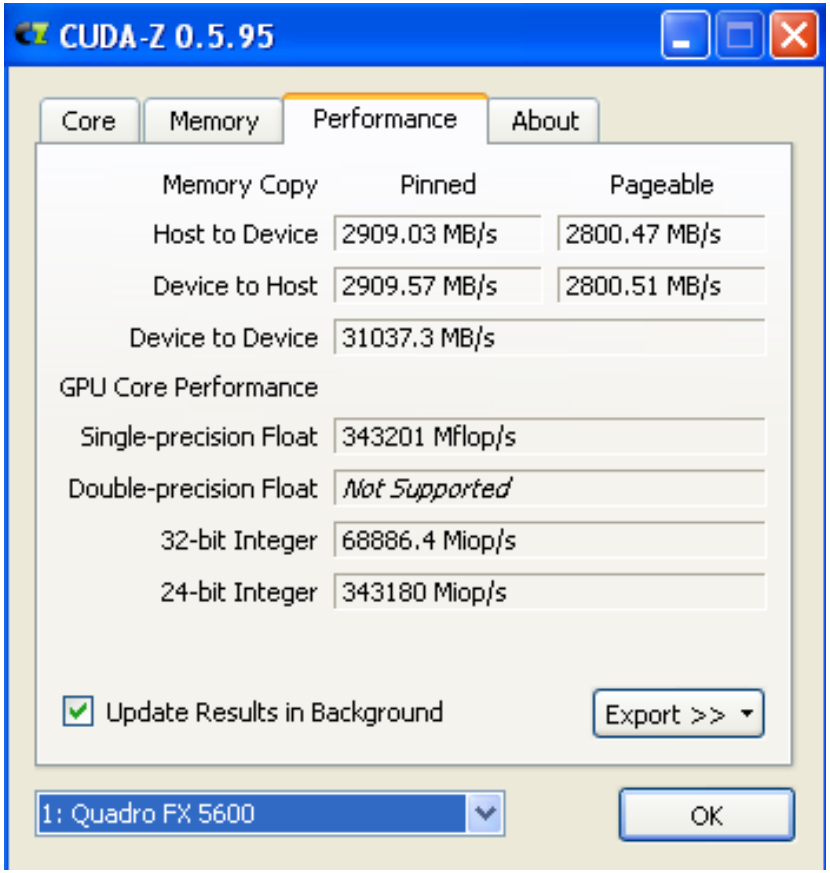
- Quadro FX 5600 & Tesla C1060

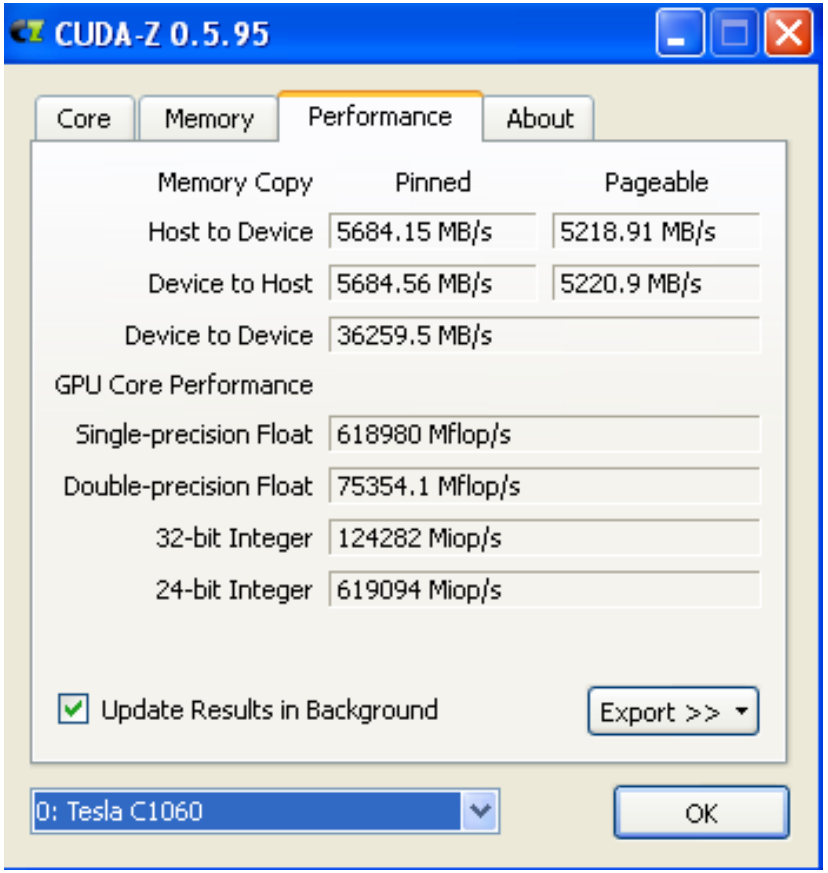




CUDA-Z

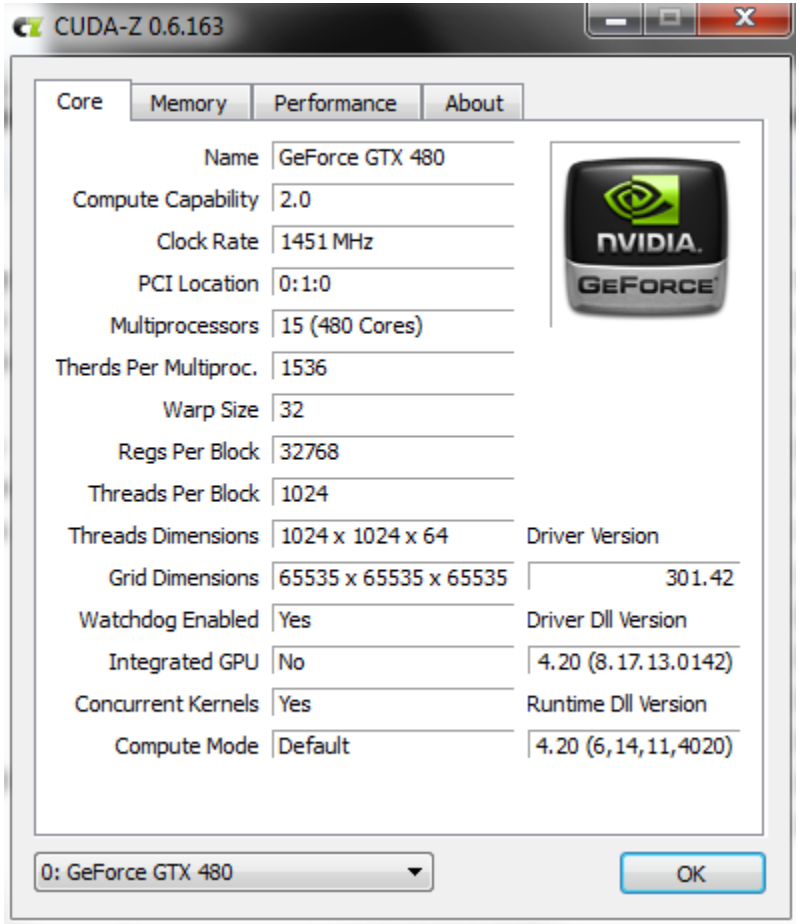
- Quadro FX 5600 & Tesla C1060

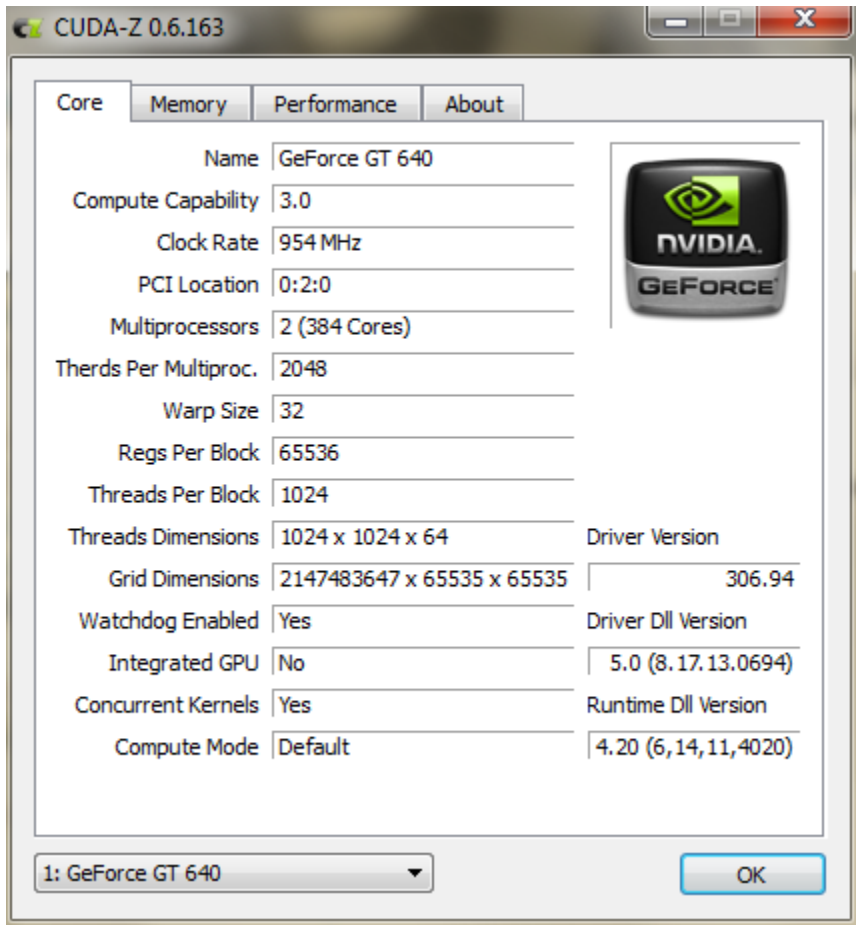




CUDA-Z

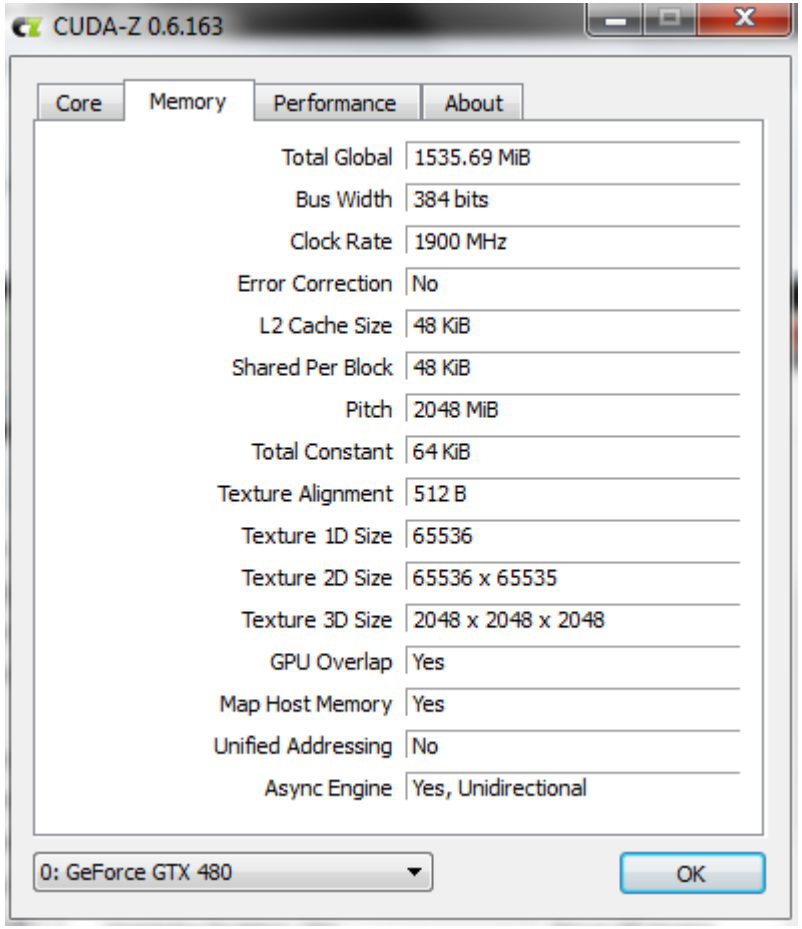
- GeForce GTX 480 & GeForce GT 640

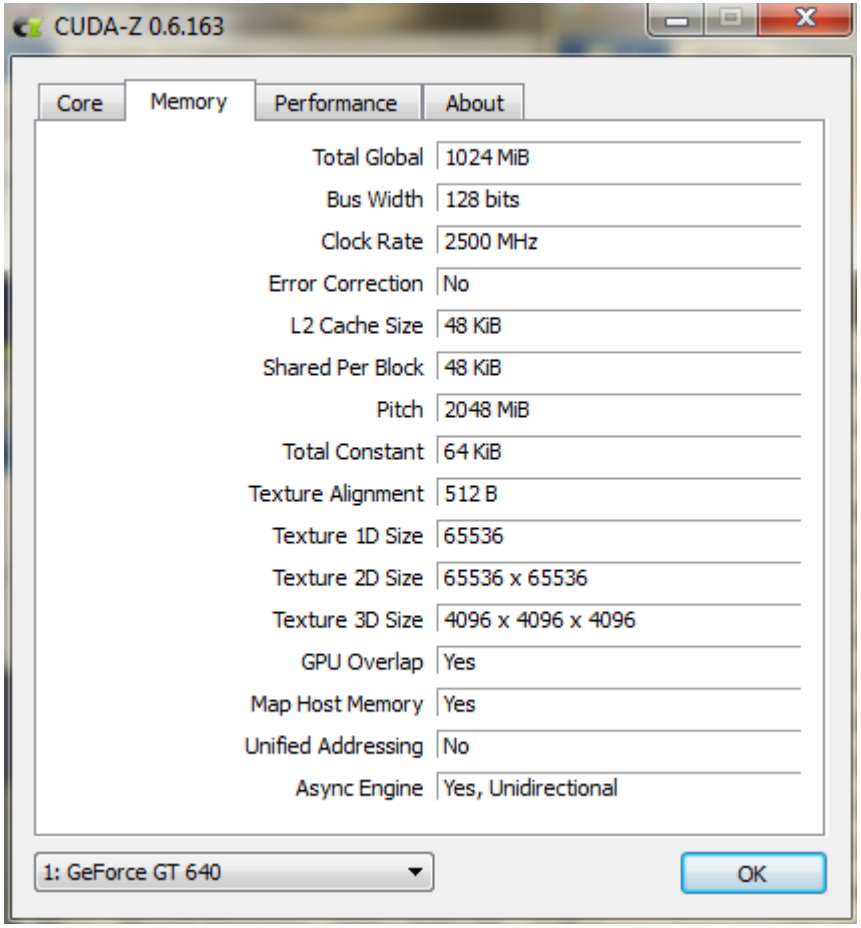




CUDA-Z

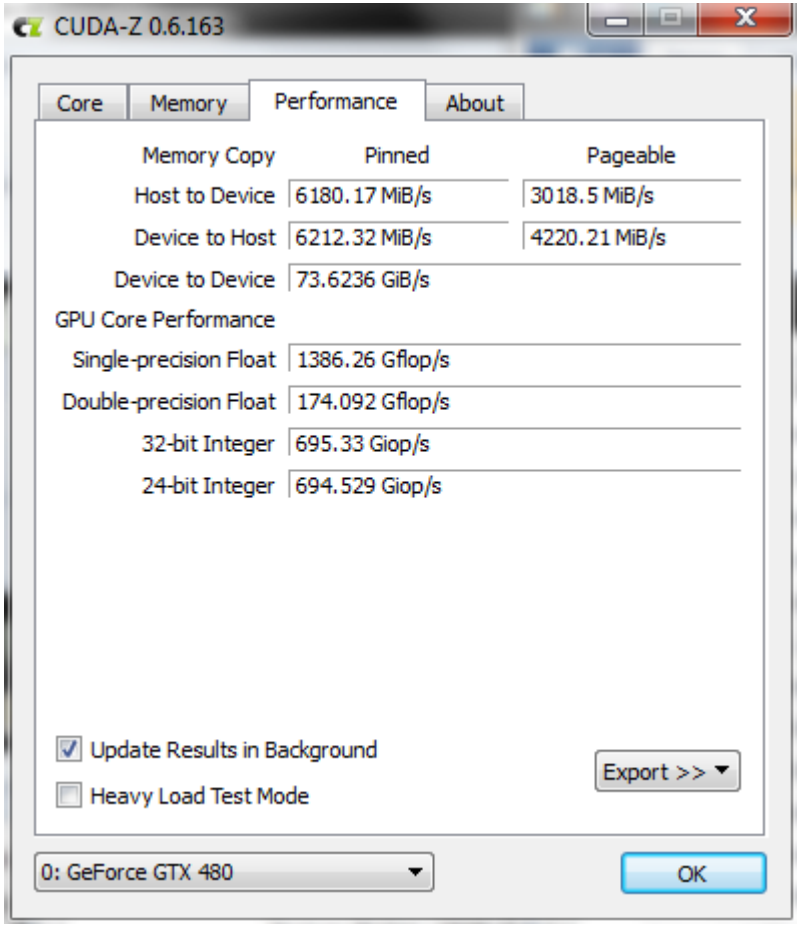
- GeForce GTX 480 & GeForce GT 640

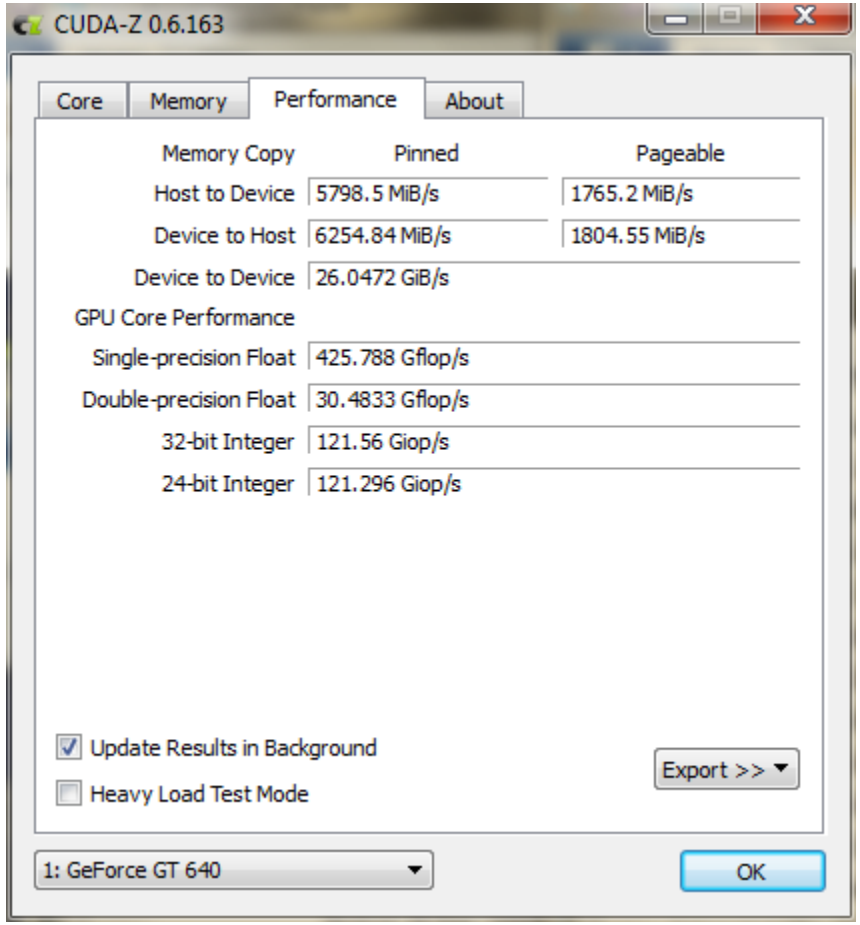




CUDA-Z

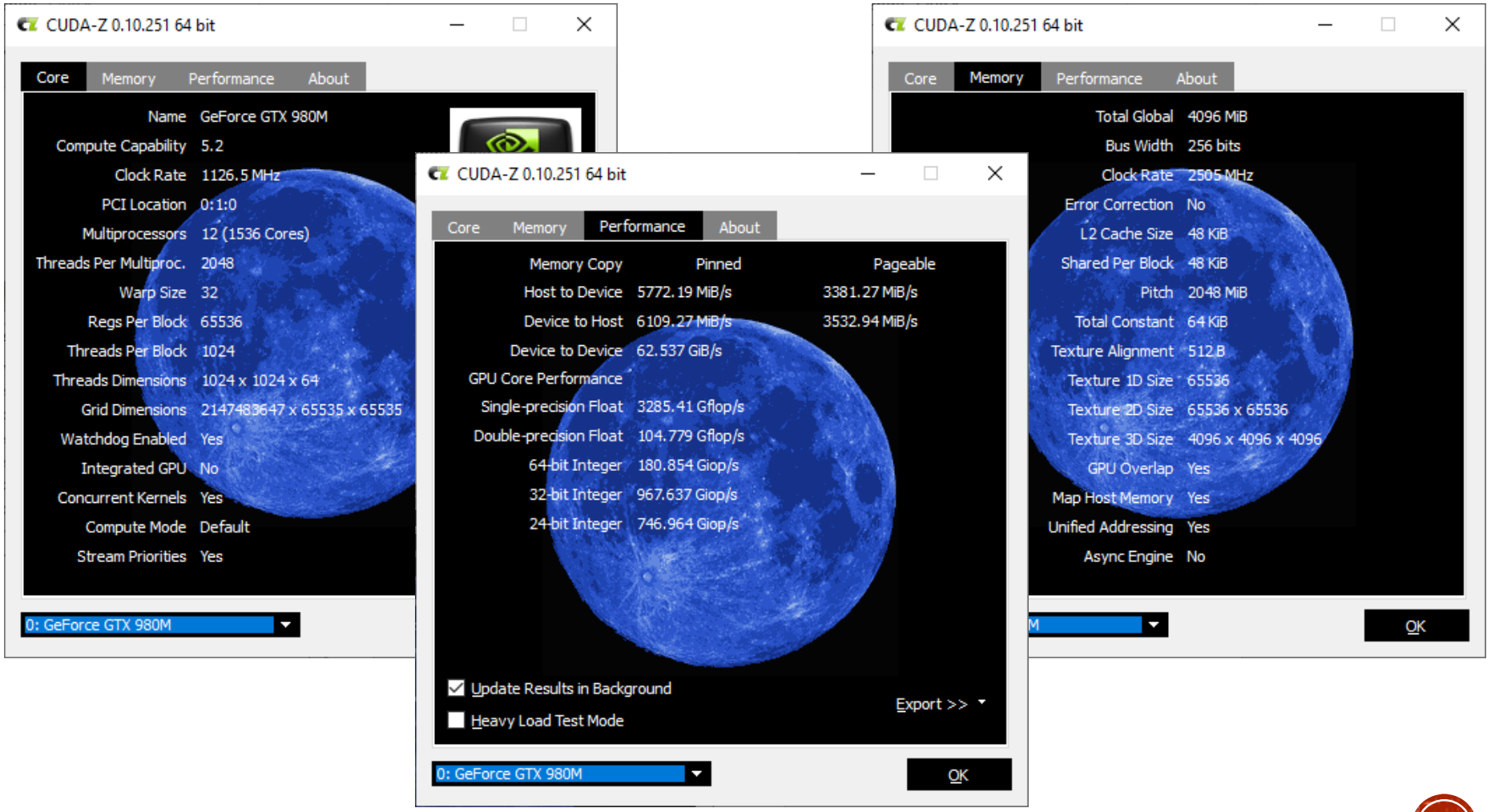
- GeForce GTX 480 & GeForce GT 640





CUDA-Z

- GeForce GTX 980M



“deviceQueryDrv”

```
Device 0: "Tesla K40c"
  CUDA Driver Version / Runtime Version          11.2 / 11.2
  CUDA Capability Major/Minor version number:    3.5
  Total amount of global memory:                 11441 MBytes (11996954624 bytes)
  (15) Multiprocessors, (192) CUDA Cores/MP:     2880 CUDA Cores
  GPU Max Clock rate:                            745 MHz (0.75 GHz)
  Memory Clock rate:                             3004 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 1572864 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 shared memory per multiprocessor:        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:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            No
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 8 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

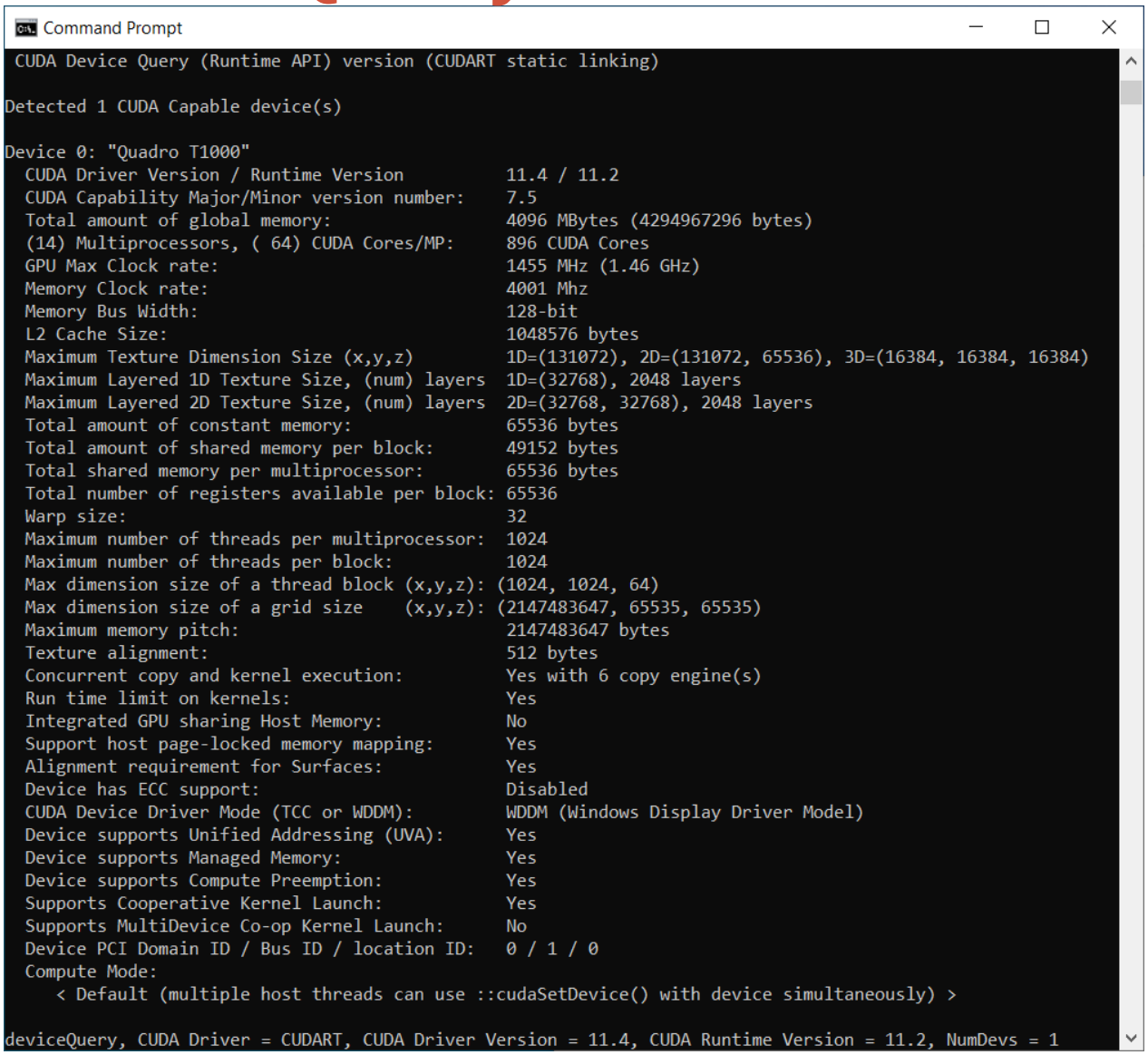
“deviceQueryDrv”

```
Device 0: "TITAN RTX"
  CUDA Driver Version / Runtime Version          11.0 / 11.0
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 24220 MBytes (25396838400 bytes)
  (72) Multiprocessors, ( 64) CUDA Cores/MP:     4608 CUDA Cores
  GPU Max Clock rate:                            1770 MHz (1.77 GHz)
  Memory Clock rate:                             7001 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 6291456 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:  1024
  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 3 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
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 175 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

“deviceQueryDrv”

```
Device 1: "Quadro RTX 8000"
  CUDA Driver Version / Runtime Version          11.0 / 11.0
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 48601 MBytes (50962169856 bytes)
  (72) Multiprocessors, ( 64) CUDA Cores/MP:     4608 CUDA Cores
  GPU Max Clock rate:                            1770 MHz (1.77 GHz)
  Memory Clock rate:                             7001 Mhz
  Memory Bus Width:                              384-bit
  L2 Cache Size:                                 6291456 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:  1024
  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 3 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
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 216 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
```

“deviceQueryDrv”



GRACIAS POR SU ATENCIÓN

Francisco J. Hernández-López

fcoj23@cimat.mx

WebPage:

www.cimat.mx/~fcoj23