WIKIPEDIA

CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia.[1] It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.[2]

The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming.[3] CUDA-powered GPUs also support programming frameworks such as OpenACC and OpenCL;[4][2] and HIP by compiling such code to CUDA. When CUDA was first introduced by Nvidia, the name was an acronym for Compute Unified Device Architecture,[5] but Nvidia subsequently dropped the common use of the acronym.

CUDA

Developer(s)	Nvidia Corporation
Initial release	June 23, 2007
Stable release	11.2.0 / December 17, 2020
Operating system	Windows, Linux
Platform	Supported GPUs
Type	GPGPU
License	Proprietary
Website	developer.nvidia.com/cuda-zone (https://developer.nvidia.com/cuda-zone)

Contents

Background
Programming abilities
Advantages
Limitations
GPUs supported
Version features and specifications
Example
Current and future usages of CUDA architecture
See also
References
External links

Background

The graphics processing unit (GPU), as a specialized computer processor, addresses the demands of real-time high-resolution 3D graphics compute-intensive tasks. By 2012, GPUs had evolved into highly parallel multi-core systems allowing very efficient manipulation of large blocks of data. This design is more effective than general-purpose central processing unit (CPUs) for algorithms in situations where processing large blocks of data is done in parallel, such as:

- push-relabel maximum flow algorithm
- fast sort algorithms of large lists
- two-dimensional fast wavelet transform
- molecular dynamics simulations
- machine learning

Programming abilities

The CUDA platform is accessible to software developers through CUDA-accelerated libraries, compiler directives such as OpenACC, and extensions to industry-standard programming languages including C, C++ and Fortran. C/C++ programmers can use 'CUDA C/C++', compiled to PTX with *nvcc*, Nvidia's LLVM-based C/C++ compiler.[6] Fortran programmers can use 'CUDA Fortran', compiled with the PGI CUDA Fortran compiler from The Portland Group.

In addition to libraries, compiler directives, CUDA C/C++ and CUDA Fortran, the CUDA platform supports other computational interfaces, including the Khronos Group's OpenCL,[7] Microsoft's DirectCompute, OpenGL Compute Shader and C++ AMP.[8] Third party wrappers are also available for Python, Perl, Fortran, Java, Ruby, Lua, Common Lisp, Haskell, R, MATLAB, IDL, Julia, and native support in Mathematica.

In the computer game industry, GPUs are used for graphics rendering, and for game physics calculations (physical effects such as debris, smoke, fire, fluids); examples include PhysX and Bullet. CUDA has also been used to accelerate non-graphical applications in computational biology, cryptography and other fields by an order of magnitude or more.[9][10][11][12][13]

CUDA provides both a low level API (CUDA **Driver** API, non single-source) and a higher level API (CUDA **Runtime** API, single-source). The initial CUDA SDK was made public on 15 February 2007, for Microsoft Windows and Linux. Mac OS X support was later added in version 2.0,[14] which supersedes the beta released February 14, 2008.[15] CUDA works with all Nvidia GPUs from the G8x series onwards, including GeForce, Quadro and the Tesla line. CUDA is compatible with most standard operating systems.

CUDA 8.0 comes with the following libraries (for compilation & runtime, in alphabetical order):

- cuBLAS – CUDA Basic Linear Algebra Subroutines library
- CUDART – CUDA Runtime library
- cuFFT – CUDA Fast Fourier Transform library
- cuRAND – CUDA Random Number Generation library

- cuSOLVER – CUDA based collection of dense and sparse direct solvers
- cuSPARSE – CUDA Sparse Matrix library
- NPP – NVIDIA Performance Primitives library
- nvGRAPH – NVIDIA Graph Analytics library
- NVML – NVIDIA Management Library
- NVRTC – NVIDIA Runtime Compilation library for CUDA C++

CUDA 8.0 comes with these other software components:

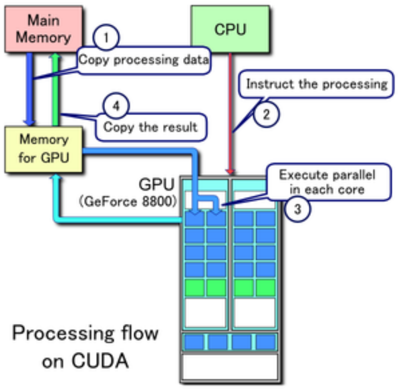
- nView – NVIDIA nView Desktop Management Software
- NVWMI – NVIDIA Enterprise Management Toolkit
- GameWorks PhysX – is a multi-platform game physics engine

CUDA 9.0–9.2 comes with these other components:

- CUTLASS 1.0 – custom linear algebra algorithms,
- ~~NVCUVID~~ – NVIDIA Video Decoder was deprecated in CUDA 9.2; it is now available in NVIDIA Video Codec SDK

CUDA 10 comes with these other components:

- nvJPEG – Hybrid (CPU and GPU) JPEG processing



Example of CUDA processing flow

1. Copy data from main memory to GPU memory
2. CPU initiates the GPU compute kernel
3. GPU's CUDA cores execute the kernel in parallel
4. Copy the resulting data from GPU memory to main memory

Advantages

CUDA has several advantages over traditional general-purpose computation on GPUs (GPGPU) using graphics APIs:

- Scattered reads – code can read from arbitrary addresses in memory.
- Unified virtual memory (CUDA 4.0 and above)
- Unified memory (CUDA 6.0 and above)
- Shared memory – CUDA exposes a fast shared memory region that can be shared among threads. This can be used as a user-managed cache, enabling higher bandwidth than is possible using texture lookups.[16]
- Faster downloads and readbacks to and from the GPU
- Full support for integer and bitwise operations, including integer texture lookups
- On RTX 20 and 30 series cards, the CUDA cores are used for a feature called "RTX IO" Which is where the CUDA cores dramatically decrease game-loading times.

Limitations

- Whether for the host computer or the GPU device, all CUDA source code is now processed according to C++ syntax rules.[17] This was not always the case. Earlier versions of CUDA were based on C syntax rules.[18] As with the more general case of compiling C code with a C++ compiler, it is therefore possible that old C-style CUDA source code will either fail to compile or will not behave as originally intended.
- Interoperability with rendering languages such as OpenGL is one-way, with OpenGL having access to registered CUDA memory but CUDA not having access to OpenGL memory.
- Copying between host and device memory may incur a performance hit due to system bus bandwidth and latency (this can be partly alleviated with asynchronous memory transfers, handled by the GPU's DMA engine).
- Threads should be running in groups of at least 32 for best performance, with total number of threads numbering in the thousands. Branches in the program code do not affect performance significantly, provided that each of 32 threads takes the same execution path; the SIMD execution model becomes a significant limitation for any inherently divergent task (e.g. traversing a space partitioning data structure during ray tracing).
- No emulator or fallback functionality is available for modern revisions.
- Valid C++ may sometimes be flagged and prevent compilation due to the way the compiler approaches optimization for target GPU device limitations.
- C++ run-time type information (RTTI) and C++-style exception handling are only supported in host code, not in device code.
- In single-precision on first generation CUDA compute capability 1.x devices, denormal numbers are unsupported and are instead flushed to zero, and the precision of both the division and square root operations are slightly lower than IEEE 754-compliant single precision math. Devices that support compute capability 2.0 and above support denormal numbers, and the division and square root operations are IEEE 754 compliant by default. However, users can obtain the prior faster gaming-grade math of compute capability 1.x devices if desired by setting compiler flags to disable accurate divisions and accurate square roots, and enable flushing denormal numbers to zero.[19]
- Unlike OpenCL, CUDA-enabled GPUs are only available from Nvidia.[20] Attempts to implement CUDA on other GPUs include:
  - Project Coriander: Converts CUDA C++11 source to OpenCL 1.2 C. A fork of CUDA-on-CL intended to run TensorFlow.[21][22][23]
  - CU2CL: Convert CUDA 3.2 C++ to OpenCL C.[24]
  - GPUOpen HIP: A thin abstraction layer on top of CUDA and ROCm intended for AMD and Nvidia GPUs. Has a conversion tool for importing CUDA C++ source. Supports CUDA 4.0 plus C++11 and float16.

GPUs supported

Supported CUDA level of GPU and card. See also at Nvidia (http://developer.nvidia.com/cuda-gpus):

- CUDA SDK 1.0 support for compute capability 1.0 – 1.1 (Tesla)[25]
- CUDA SDK 1.1 support for compute capability 1.0 – 1.1+x (Tesla)
- CUDA SDK 2.0 support for compute capability 1.0 – 1.1+x (Tesla)

- CUDA SDK 2.1 – 2.3.1 support for compute capability 1.0 – 1.3 (Tesla)[26][27][28][29]
- CUDA SDK 3.0 – 3.1 support for compute capability 1.0 – 2.0 (Tesla, Fermi)[30][31]
- CUDA SDK 3.2 support for compute capability 1.0 – 2.1 (Tesla, Fermi)[32]
- CUDA SDK 4.0 – 4.2 support for compute capability 1.0 – 2.1+x (Tesla, Fermi, more?).
- CUDA SDK 5.0 – 5.5 support for compute capability 1.0 – 3.5 (Tesla, Fermi, Kepler).
- CUDA SDK 6.0 support for compute capability 1.0 – 3.5 (Tesla, Fermi, Kepler).
- CUDA SDK 6.5 support for compute capability 1.1 – 5.x (Tesla, Fermi, Kepler, Maxwell). Last version with support for compute capability 1.x (Tesla)
- CUDA SDK 7.0 – 7.5 support for compute capability 2.0 – 5.x (Fermi, Kepler, Maxwell).
- CUDA SDK 8.0 support for compute capability 2.0 – 6.x (Fermi, Kepler, Maxwell, Pascal). Last version with support for compute capability 2.x (Fermi) (Pascal GTX 1070Ti Not Supported)
- CUDA SDK 9.0 – 9.2 support for compute capability 3.0 – 7.2 (Kepler, Maxwell, Pascal, Volta) (Pascal GTX 1070Ti Not Supported. CUDA SDK 9.0 and support CUDA SDK 9.2).
- CUDA SDK 10.0 – 10.2 support for compute capability 3.0 – 7.5 (Kepler, Maxwell, Pascal, Volta, Turing). Last version with support for compute capability 3.x (Kepler). 10.2 is the last official release for macOS, as support will not be available for macOS in newer releases.
- CUDA SDK 11.0 – 11.2 support for compute capability 3.5 – 8.6 (Kepler (in part), Maxwell, Pascal, Volta, Turing, Ampere)[33] New data types: Bfloat16 and TF32 on third-generations Tensor Cores.[34]

Compute capability (version)	Micro-architecture	GPUs	GeForce	Quadro, NVS	Tesla	Tegra, Jetson, DRIVE
1.0	Tesla	G80	GeForce 8800 Ultra, GeForce 8800 GTX, GeForce 8800 GTS(G80)	Quadro FX 5600, Quadro FX 4600, Quadro Plex 2100 S4	Tesla C870, Tesla D870, Tesla S870	
1.1		G92, G94, G96, G98, G84, G86	GeForce GTS 250, GeForce 9800 GX2, GeForce 9800 GTX, GeForce 9800 GT, GeForce 8800 GTS(G92), GeForce 8800 GT, GeForce 9600 GT, GeForce 9500 GT, GeForce 9400 GT, GeForce 8600 GTS, GeForce 8600 GT, GeForce 8500 GT, GeForce G110M, GeForce 9300M GS, GeForce 9200M GS, GeForce 9100M G, GeForce 8400M GT, GeForce G105M	Quadro FX 4700 X2, Quadro FX 3700, Quadro FX 1800, Quadro FX 1700, Quadro FX 580, Quadro FX 570, Quadro FX 470, Quadro FX 380, Quadro FX 370, Quadro FX 370 Low Profile, Quadro NVS 450, Quadro NVS 420, Quadro NVS 290, Quadro NVS 295, Quadro Plex 2100 D4, Quadro FX 3800M, Quadro FX 3700M, Quadro FX 3600M, Quadro FX 2800M, Quadro FX 2700M, Quadro FX 1700M, Quadro FX 1600M, Quadro FX 770M, Quadro FX 570M, Quadro FX 370M, Quadro FX 360M, Quadro NVS 320M, Quadro NVS 160M, Quadro NVS 150M, Quadro NVS 140M, Quadro NVS 135M, Quadro NVS 130M, Quadro NVS 450, Quadro NVS 420,[35] Quadro NVS 295		
1.2		GT218, GT216, GT215	GeForce GT 340*, GeForce GT 330*, GeForce GT 320*, GeForce 315*, GeForce 310*, GeForce GT 240, GeForce GT 220, GeForce 210, GeForce GTS 360M, GeForce GTS 350M, GeForce GT 335M, GeForce GT 330M, GeForce GT 325M, GeForce GT 240M, GeForce G210M, GeForce 310M, GeForce 305M	Quadro FX 380 Low Profile, Quadro FX 1800M, Quadro FX 880M, Quadro FX 380M, Nvidia NVS 300, NVS 5100M, NVS 3100M, NVS 2100M, ION		
1.3		GT200, GT200b	GeForce GTX 295, GTX 285, GTX 280, GeForce GTX 275, GeForce GTX 260	Quadro FX 5800, Quadro FX 4800, Quadro FX 4800 for Mac, Quadro FX 3800, Quadro CX, Quadro Plex 2200 D2	Tesla C1060, Tesla S1070, Tesla M1060	
2.0	Fermi	GF100, GF110	GeForce GTX 590, GeForce GTX 580, GeForce GTX 570, GeForce GTX 480, GeForce GTX 470, GeForce GTX 465, GeForce GTX 480M	Quadro 6000, Quadro 5000, Quadro 4000, Quadro 4000 for Mac, Quadro Plex 7000, Quadro 5010M, Quadro 5000M	Tesla C2075, Tesla C2050/C2070, Tesla M2050/M2070/M2075/M2090	
2.1		GF104, GF106 GF108, GF114, GF116, GF117, GF119	GeForce GTX 560 Ti, GeForce GTX 550 Ti, GeForce GTX 460, GeForce GTS 450, GeForce GTS 450*, GeForce GT 640 (GDDR3), GeForce GT 630, GeForce GT 620, GeForce GT 610, GeForce GT 520, GeForce GT 440, GeForce GT 440*, GeForce GT 430, GeForce GT 430*, GeForce GT 420*, GeForce GTX 675M, GeForce GTX 670M, GeForce GT 635M, GeForce GT 630M, GeForce GT 625M, GeForce GT 720M, GeForce GT 620M, GeForce 710M, GeForce 610M, GeForce 820M, GeForce GTX 580M, GeForce GTX 570M, GeForce GTX 560M, GeForce GT 555M, GeForce GT 550M, GeForce GT 540M, GeForce GT 525M, GeForce GT 520MX, GeForce GT 520M, GeForce GTX 485M, GeForce GTX 470M, GeForce GTX 460M, GeForce GT 445M, GeForce GT 435M, GeForce GT 420M, GeForce GT 415M, GeForce 710M, GeForce 410M	Quadro 2000, Quadro 2000D, Quadro 600, Quadro 4000M, Quadro 3000M, Quadro 2000M, Quadro 1000M, NVS 310, NVS 315, NVS 5400M, NVS 5200M, NVS 4200M		
3.0	Kepler	GK104, GK106, GK107	GeForce GTX 770, GeForce GTX 760, GeForce GT 740, GeForce GTX 690, GeForce GTX 680, GeForce GTX 670, GeForce GTX 660 Ti, GeForce GTX 660, GeForce GTX 650 Ti BOOST, GeForce GTX 650 Ti, GeForce GTX 650, GeForce GTX 880M, GeForce GTX 870M, GeForce GTX 780M, GeForce GTX 770M, GeForce GTX 765M, GeForce GTX 760M, GeForce GTX 680MX, GeForce GTX 680M, GeForce GTX 675MX, GeForce GTX 670MX, GeForce GTX 660M, GeForce GT 750M, GeForce GT 650M, GeForce GT 745M, GeForce GT 645M, GeForce GT 740M, GeForce GT 730M, GeForce GT 640M, GeForce GT 640M LE,	Quadro K5000, Quadro K4200, Quadro K4000, Quadro K2000, Quadro K2000D, Quadro K600, Quadro K420, Quadro K500M, Quadro K510M, Quadro K610M, Quadro K1000M, Quadro K2000M, Quadro K1100M, Quadro K2100M, Quadro K3000M, Quadro K3100M, Quadro K4000M, Quadro K5000M, Quadro K4100M, Quadro K5100M, NVS 510, Quadro 410	Tesla K10, GRID K340, GRID K520, GRID K2	

			GeForce GT 735M, GeForce GT 730M			
3.2		GK20A				Tegra K1, Jetson TK1
3.5		GK110, GK208	GeForce GTX Titan Z, GeForce GTX Titan Black, GeForce GTX Titan, GeForce GTX 780 Ti, GeForce GTX 780, GeForce GT 640 (GDDR5), GeForce GT 630 v2, GeForce GT 730, GeForce GT 720, GeForce GT 710, GeForce GT 740M (64-bit, DDR3), GeForce GT 920M	Quadro K6000, Quadro K5200	Tesla K40, Tesla K20x, Tesla K20	
3.7		GK210			Tesla K80	
5.0	Maxwell	GM107, GM108	GeForce GTX 750 Ti, GeForce GTX 750, GeForce GTX 960M, GeForce GTX 950M, GeForce 940M, GeForce 930M, GeForce GTX 860M, GeForce GTX 850M, GeForce 845M, GeForce 840M, GeForce 830M	Quadro K1200, Quadro K2200, Quadro K620, Quadro M2000M, Quadro M1000M, Quadro M600M, Quadro K620M, NVS 810	Tesla M10	
5.2		GM200, GM204, GM206	GeForce GTX Titan X, GeForce GTX 980 Ti, GeForce GTX 980, GeForce GTX 970, GeForce GTX 960, GeForce GTX 950, GeForce GTX 750 SE, GeForce GTX 980M, GeForce GTX 970M, GeForce GTX 965M	Quadro M6000 24GB, Quadro M6000, Quadro M5000, Quadro M4000, Quadro M2000, Quadro M5500, Quadro M5000M, Quadro M4000M, Quadro M3000M	Tesla M4, Tesla M40, Tesla M6, Tesla M60	
5.3		GM20B				Tegra X1, Jetson TX1, Jetson Nano, DRIVE CX, DRIVE PX
6.0	Pascal	GP100		Quadro GP100	Tesla P100	
6.1		GP102, GP104, GP106, GP107, GP108	Nvidia TITAN Xp, Titan X, GeForce GTX 1080 Ti, GTX 1080, GTX 1070 Ti, GTX 1070, GTX 1060, GTX 1050 Ti, GTX 1050, GT 1030, GT 1010 MX350, MX330, MX250, MX230, MX150, MX130, MX110	Quadro P6000, Quadro P5000, Quadro P4000, Quadro P2200, Quadro P2000, Quadro P1000, Quadro P400, Quadro P500, Quadro P520, Quadro P600, Quadro P5000(Mobile), Quadro P4000(Mobile), Quadro P3000(Mobile)	Tesla P40, Tesla P6, Tesla P4	
6.2		GP10B[36]				Tegra X2, Jetson TX2, DRIVE PX 2
7.0	Volta	GV100	NVIDIA TITAN V	Quadro GV100	Tesla V100, Tesla V100S	
7.2		GV10B[37]				Tegra Xavier, Jetson Xavier NX, Jetson AGX Xavier, DRIVE AGX Xavier, DRIVE AGX Pegasus
7.5	Turing	TU102, TU104, TU106, TU116, TU117	NVIDIA TITAN RTX, GeForce RTX 2080 Ti, RTX 2080 Super, RTX 2080, RTX 2070 Super, RTX 2070, RTX 2060 Super, RTX 2060, GeForce GTX 1660 Ti, GTX 1660 Super, GTX 1660, GTX 1650 Super, GTX 1650, MX450	Quadro RTX 8000, Quadro RTX 6000, Quadro RTX 5000, Quadro RTX 4000, Quadro T2000, Quadro T1000	Tesla T4	
8.0	Ampere	GA100			A100 80GB, A100 40GB	
8.6		GA102, GA104, GA106	GeForce RTX 3090, RTX 3080, RTX 3070, RTX 3060 Ti, RTX 3060, RTX 3050 Ti	RTX A6000, A40		

'*' – OEM-only products

Version features and specifications

<table>
<tr><th rowspan="2">Feature support (unlisted features are supported for all compute capabilities)</th><th colspan="13">Compute capability (version)</th></tr>
<tr><th>1.0</th><th>1.1</th><th>1.2</th><th>1.3</th><th>2.x</th><th>3.0</th><th>3.2</th><th>3.5, 3.7, 5.0, 5.2</th><th>5.3</th><th>6.x</th><th>7.x</th><th>8.0</th><th>8.6</th></tr>
<tr><td>Integer atomic functions operating on 32-bit words in global memory</td><td rowspan="2">No</td><td rowspan="2" colspan="12">Yes</td></tr>
<tr><td>atomicExch() operating on 32-bit floating point values in global memory</td></tr>
<tr><td>Integer atomic functions operating on 32-bit words in shared memory</td><td rowspan="4" colspan="2">No</td><td rowspan="4" colspan="11">Yes</td></tr>
<tr><td>atomicExch() operating on 32-bit floating point values in shared memory</td></tr>
<tr><td>Integer atomic functions operating on 64-bit words in global memory</td></tr>
<tr><td>Warp vote functions</td></tr>
<tr><td>Double-precision floating-point operations</td><td colspan="3">No</td><td colspan="10">Yes</td></tr>
<tr><td>Atomic functions operating on 64-bit integer values in shared memory</td><td rowspan="7" colspan="4">No</td><td rowspan="7" colspan="9">Yes</td></tr>
<tr><td>Floating-point atomic addition operating on 32-bit words in global and shared memory</td></tr>
<tr><td>_ballot()</td></tr>
<tr><td>_threadfence_system()</td></tr>
<tr><td>_syncthreads_count(), _syncthreads_and(), _syncthreads_or()</td></tr>
<tr><td>Surface functions</td></tr>
<tr><td>3D grid of thread block</td></tr>
<tr><td>Warp shuffle functions , Unified Memory</td><td colspan="5">No</td><td colspan="8">Yes</td></tr>
<tr><td>Funnel shift</td><td colspan="6">No</td><td colspan="7">Yes</td></tr>
<tr><td>Dynamic parallelism</td><td colspan="7">No</td><td colspan="6">Yes</td></tr>
<tr><td>Half-precision floating-point operations:
addition, subtraction, multiplication, comparison, warp shuffle functions, conversion</td><td colspan="8">No</td><td colspan="5">Yes</td></tr>
<tr><td>Atomic addition operating on 64-bit floating point values in global memory and shared memory</td><td colspan="9">No</td><td colspan="4">Yes</td></tr>
<tr><td>Tensor core</td><td colspan="10">No</td><td colspan="3">Yes</td></tr>
<tr><td>Mixed Precision Warp-Matrix Functions</td><td colspan="10">No</td><td colspan="3">Yes</td></tr>
<tr><td>Hardware-accelerated async-copy</td><td colspan="11">No</td><td colspan="2">Yes</td></tr>
<tr><td>Hardware-accelerated Split Arrive/Wait Barrier</td><td colspan="11">No</td><td colspan="2">Yes</td></tr>
<tr><td>L2 Cache Residency Management</td><td colspan="11">No</td><td colspan="2">Yes</td></tr>
</table>

[38]

Data Type	Operation	Supported since	Supported since for global memory	Supported since for shared memory
16-bit integer	general operations			
32-bit integer	atomic functions		1.1	1.2
64-bit integer	atomic functions		1.2	2.0
16-bit floating point	addition, subtraction, multiplication, comparison, warp shuffle functions, conversion	5.3		
32-bit floating point	atomicExch()		1.1	1.2
32-bit floating point	atomic addition		2.0	2.0
64-bit floating point	general operations	1.3		
64-bit floating point	atomic addition		6.0	6.0
	tensor core	7.0		

Note: Any missing lines or empty entries do reflect some lack of information on that exact item.
[39]

Technical specifications	Compute capability (version)																			
	1.0	**1.1**	**1.2**	**1.3**	**2.x**	**3.0**	**3.2**	**3.5**	**3.7**	**5.0**	**5.2**	**5.3**	**6.0**	**6.1**	**6.2**	**7.0**	**7.2**	**7.5**	**8.0**	**8.6**
Maximum number of resident grids per device (concurrent kernel execution)	t.b.d.				16		4	32				16	128	32	16	128	16	128		
Maximum dimensionality of grid of thread blocks	2				3															
Maximum x-dimension of a grid of thread blocks	65535					$2^{31} - 1$														
Maximum y-, or z-dimension of a grid of thread blocks	65535																			
Maximum dimensionality of thread block	3																			
Maximum x- or y-dimension of a block	512				1024															
Maximum z-dimension of a block	64																			
Maximum number of threads per block	512				1024															
Warp size	32																			
Maximum number of resident blocks per multiprocessor	8					16				32								16	32	16
Maximum number of resident warps per multiprocessor	24		32		48	64												32	64	48
Maximum number of resident threads per multiprocessor	768		1024		1536	2048												1024	2048	153
Number of 32-bit registers per multiprocessor	8 K		16 K		32 K	64 K			128 K	64 K										
Maximum number of 32-bit registers per thread block	N/A				32 K	64 K	32 K	64 K				32 K	64 K		32 K	64 K				
Maximum number of 32-bit registers per thread	124				63		255													
Maximum amount of shared memory per multiprocessor	16 KB				48 KB				112 KB	64 KB	96 KB	64 KB		96 KB	64 KB	96 KB (of 128)		64 KB (of 96)	164 KB (of 192)	100 K (of 12
Maximum amount of shared memory per thread block	48 KB															96 KB	48 KB	64 KB	163 KB	99 K
Number of shared memory banks	16				32															
Amount of local memory per thread	16 KB				512 KB															
Constant memory size	64 KB																			
Cache working set per	8 KB												4 KB	8 KB						

multiprocessor for constant memory															
Cache working set per multiprocessor for texture memory	6 – 8 KB	12 KB		12 – 48 KB	24 KB	48 KB	N/A	24 KB	48 KB	24 KB	32 – 128 KB		32 – 64 KB	28 – 192 KB	28 – 128 K
Maximum width for 1D texture reference bound to a CUDA array	8192	65536						131072							
Maximum width for 1D texture reference bound to linear memory	2^{27}							2^{28}	2^{27}		2^{28}	2^{27}	2^{28}		
Maximum width and number of layers for a 1D layered texture reference	8192 × 512	16384 × 2048						32768 x 2048							
Maximum width and height for 2D texture reference bound to a CUDA array	65536 × 32768	65536 × 65535						131072 x 65536							
Maximum width and height for 2D texture reference bound to a linear memory	65000 x 65000				65536 x 65536			131072 x 65000							
Maximum width and height for 2D texture reference bound to a CUDA array supporting texture gather	N/A	16384 x 16384						32768 x 32768							
Maximum width, height, and number of layers for a 2D layered texture reference	8192 × 8192 × 512	16384 × 16384 × 2048						32768 x 32768 x 2048							
Maximum width, height and depth for a 3D texture reference bound to linear memory or a CUDA array	2048^3		4096^3					16384^3							
Maximum width (and height) for a cubemap texture reference	N/A	16384						32768							
Maximum width (and height) and number of layers for a cubemap layered texture reference	N/A	16384 × 2046						32768 × 2046							
Maximum number of textures that can be bound to a kernel	128		256												
Maximum width for a 1D	Not supported	65536			16384			32768							

<table>
<tr><td>surface reference bound to a CUDA array</td><td rowspan="8"></td><td colspan="2"></td><td></td><td colspan="2"></td></tr>
<tr><td>Maximum width and number of layers for a 1D layered surface reference</td><td colspan="2">65536 × 2048</td><td>16384 × 2048</td><td colspan="2">32768 × 2048</td></tr>
<tr><td>Maximum width and height for a 2D surface reference bound to a CUDA array</td><td colspan="2">65536 × 32768</td><td>16384 × 65536</td><td colspan="2">131072 × 65536</td></tr>
<tr><td>Maximum width, height, and number of layers for a 2D layered surface reference</td><td colspan="2">65536 × 32768 × 2048</td><td>16384 × 16384 × 2048</td><td colspan="2">32768 × 32768 × 2048</td></tr>
<tr><td>Maximum width, height, and depth for a 3D surface reference bound to a CUDA array</td><td colspan="2">65536 × 32768 × 2048</td><td>4096 × 4096 × 4096</td><td colspan="2">16384 × 16384 × 16384</td></tr>
<tr><td>Maximum width (and height) for a cubemap surface reference bound to a CUDA array</td><td colspan="2">32768</td><td>16384</td><td colspan="2">32768</td></tr>
<tr><td>Maximum width and number of layers for a cubemap layered surface reference</td><td colspan="2">32768 × 2046</td><td>16384 × 2046</td><td colspan="2">32768 × 2046</td></tr>
<tr><td>Maximum number of surfaces that can be bound to a kernel</td><td>8</td><td colspan="3">16</td><td>32</td></tr>
<tr><td>Maximum number of instructions per kernel</td><td>2 million</td><td colspan="5">512 million</td></tr>
</table>

[40]

<table>
<tr><th rowspan="2">Architecture specifications</th><th colspan="17">Compute capability (version)</th></tr>
<tr><th>1.0</th><th>1.1</th><th>1.2</th><th>1.3</th><th>2.0</th><th>2.1</th><th>3.0</th><th>3.5</th><th>3.7</th><th>5.0</th><th>5.2</th><th>6.0</th><th>6.1, 6.2</th><th>7.0, 7.2</th><th>7.5</th><th>8.0</th><th>8.6</th></tr>
<tr><td>Number of ALU lanes for integer and single-precision floating-point arithmetic operations</td><td colspan="4">8[41]</td><td>32</td><td>48</td><td colspan="3">192</td><td colspan="2">128</td><td>64</td><td>128</td><td colspan="3">64</td><td></td></tr>
<tr><td>Number of special function units for single-precision floating-point transcendental functions</td><td colspan="4">2</td><td>4</td><td>8</td><td colspan="5">32</td><td>16</td><td>32</td><td colspan="4">16</td></tr>
<tr><td>Number of texture filtering units for every texture address unit or render output unit (ROP)</td><td colspan="4">2</td><td>4</td><td>8</td><td colspan="3">16</td><td colspan="6">8[42]</td><td colspan="2"></td></tr>
<tr><td>Number of warp schedulers</td><td colspan="4">1</td><td colspan="2">2</td><td colspan="5">4</td><td>2</td><td colspan="5">4</td></tr>
<tr><td>Max number of instructions issued at once by a single scheduler</td><td colspan="5">1</td><td colspan="4">2[43]</td><td colspan="8">1</td></tr>
<tr><td>Number of tensor cores</td><td colspan="13">N/A</td><td colspan="2">8[42]</td><td colspan="2">4</td></tr>
<tr><td>Size in KB of unified memory for data cache and shared memory per multi processor</td><td colspan="13">t.b.d.</td><td>128</td><td>96[44]</td><td>192</td><td>128</td></tr>
</table>

[45]

For more information see the article: "NVIDIA CUDA Compute Capability Comparative Table" (https://www.geeks3d.com/20100606/gpu-computing-nvidia-cuda-compute-capability-comparative-table/) and read Nvidia CUDA programming guide.[46]

Example

This example code in C++ loads a texture from an image into an array on the GPU:

```
texture<float, 2, cudaReadModeElementType> tex;

void foo()
{
  cudaArray* cu_array;

  // Allocate array
  cudaChannelFormatDesc description = cudaCreateChannelDesc<float>();
  cudaMallocArray(&cu_array, &description, width, height);

  // Copy image data to array
  cudaMemcpyToArray(cu_array, image, width*height*sizeof(float), cudaMemcpyHostToDevice);

  // Set texture parameters (default)
  tex.addressMode[0] = cudaAddressModeClamp;
  tex.addressMode[1] = cudaAddressModeClamp;
  tex.filterMode = cudaFilterModePoint;
  tex.normalized = false; // do not normalize coordinates

  // Bind the array to the texture
  cudaBindTextureToArray(tex, cu_array);

  // Run kernel
  dim3 blockDim(16, 16, 1);
  dim3 gridDim((width + blockDim.x - 1)/ blockDim.x, (height + blockDim.y - 1) / blockDim.y, 1);
  kernel<<< gridDim, blockDim, 0 >>>(d_data, height, width);

  // Unbind the array from the texture
  cudaUnbindTexture(tex);
} //end foo()

__global__ void kernel(float* odata, int height, int width)
{
   unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
   unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
   if (x < width && y < height) {
      float c = tex2D(tex, x, y);
      odata[y*width+x] = c;
   }
}
```

Below is an example given in Python that computes the product of two arrays on the GPU. The unofficial Python language bindings can be obtained from *PyCUDA*.[47]

```
import pycuda.compiler as comp
import pycuda.driver as drv
import numpy
import pycuda.autoinit

mod = comp.SourceModule(
    """
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
"""
)

multiply_them = mod.get_function("multiply_them")

a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)

dest = numpy.zeros_like(a)
multiply_them(drv.Out(dest), drv.In(a), drv.In(b), block=(400, 1, 1))

print(dest - a * b)
```

Additional Python bindings to simplify matrix multiplication operations can be found in the program *pycublas*.[48]

```

import numpy
from pycublas import CUBLASMatrix

A = CUBLASMatrix(numpy.mat([[1, 2, 3], [4, 5, 6]], numpy.float32))
B = CUBLASMatrix(numpy.mat([[2, 3], [4, 5], [6, 7]], numpy.float32))
C = A * B
print(C.np_mat())
```

while CuPy directly replaces NumPy:[49]

```
import cupy

a = cupy.random.randn(400)
b = cupy.random.randn(400)

dest = cupy.zeros_like(a)

print(dest - a * b)
```

Current and future usages of CUDA architecture

- Accelerated rendering of 3D graphics
- Accelerated interconversion of video file formats
- Accelerated encryption, decryption and compression

- Bioinformatics (https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/catalog/?product_category_id=26,44,130,215,240,260,353,354,355,356,357,358,359,360,361,363,419,450,481,500,506,419&search=), e.g. NGS DNA sequencing BarraCUDA[50]
- Distributed calculations, such as predicting the native conformation of proteins
- Medical analysis simulations, for example virtual reality based on CT and MRI scan images
- Physical simulations,[51] in particular in fluid dynamics
- Neural network training in machine learning problems
- Face recognition
- Distributed computing
- Molecular dynamics
- Mining cryptocurrencies
- BOINC SETI@home
- Structure from motion (SfM) software

See also

- OpenCL – an open standard from Khronos Group for programming a variety of platforms, including GPUs, similar to lower-level CUDA **Driver** API (*non single-source*)
- SYCL – an open standard from Khronos Group for programming a variety of platforms, including GPUs, with *single-source* modern C++, similar to higher-level CUDA **Runtime** API (*single-source*)
- BrookGPU – the Stanford University graphics group's compiler
- HIP
- Array programming
- Parallel computing
- Stream processing
- rCUDA – an API for computing on remote computers
- Molecular modeling on GPU
- Vulkan , low-level, high-performance 3D graphics and computing API
- OptiX, ray tracing API by NVIDIA

References

1. "Nvidia CUDA Home Page" (https://developer.nvidia.com/cuda-zone).
2. Abi-Chahla, Fedy (June 18, 2008). "Nvidia's CUDA: The End of the CPU?" (https://www.tomshardware.com/reviews/nvidia-cuda-gpu,1954.html). Tom's Hardware. Retrieved May 17, 2015.
3. Zunitch, Peter (2018-01-24). "CUDA vs. OpenCL vs. OpenGL" (https://www.videomaker.com/article/c15/19313-cuda-vs-opencl-vs-opengl). *Videomaker*. Retrieved 2018-09-16.
4. "OpenCL" (https://developer.nvidia.com/opencl). *NVIDIA Developer*. 2013-04-24. Retrieved 2019-11-04.
5. Shimpi, Anand Lal; Wilson, Derek (November 8, 2006). "Nvidia's GeForce 8800 (G80): GPUs Re-architected for DirectX 10" (https://www.anandtech.com/show/2116/8). AnandTech. Retrieved May 16, 2015.
6. "CUDA LLVM Compiler" (https://developer.nvidia.com/cuda-llvm-compiler).
7. First OpenCL demo on a GPU (https://www.youtube.com/watch?v=r1sN1ELJfNo) on YouTube
8. DirectCompute Ocean Demo Running on Nvidia CUDA-enabled GPU (https://www.youtube.com/watch?v=K1I4kts5mqc) on YouTube
9. Vasiliadis, Giorgos; Antonatos, Spiros; Polychronakis, Michalis; Markatos, Evangelos P.; Ioannidis, Sotiris (September 2008). "Gnort: High Performance Network Intrusion Detection Using Graphics Processors" (http://www.ics.forth.gr/dcs/Activities/papers/gnort.raid08.pdf) (PDF). *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (RAID)*.
10. Schatz, Michael C.; Trapnell, Cole; Delcher, Arthur L.; Varshney, Amitabh (2007). "High-throughput sequence alignment using Graphics Processing Units" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2222658). *BMC Bioinformatics*. **8**: 474. doi:10.1186/1471-2105-8-474 (https://doi.org/10.1186%2F1471-2105-8-474). PMC 2222658 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2222658). PMID 18070356 (https://pubmed.ncbi.nlm.nih.gov/18070356).
11. Manavski, Svetlin A.; Giorgio, Valle (2008). "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2323659). *BMC Bioinformatics*. **10**: S10. doi:10.1186/1471-2105-9-S2-S10 (https://doi.org/10.1186%2F1471-2105-9-S2-S10). PMC 2323659 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2323659). PMID 18387198 (https://pubmed.ncbi.nlm.nih.gov/18387198).
12. "Pyrit – Google Code" (https://code.google.com/p/pyrit/).
13. "Use your Nvidia GPU for scientific computing" (https://web.archive.org/web/20081228022142/http://boinc.berkeley.edu/cuda.php). BOINC. 2008-12-18. Archived from the original (http://boinc.berkeley.edu/cuda.php) on 2008-12-28. Retrieved 2017-08-08.
14. "Nvidia CUDA Software Development Kit (CUDA SDK) – Release Notes Version 2.0 for MAC OS X" (https://web.archive.org/web/20090106020401/http://developer.download.nvidia.com/compute/cuda/sdk/website/doc/CUDA_SDK_release_notes_macosx.txt). Archived from the original (http://developer.download.nvidia.com/compute/cuda/sdk/website/doc/CUDA_SDK_release_notes_macosx.txt) on 2009-01-06.
15. "CUDA 1.1 – Now on Mac OS X" (https://web.archive.org/web/20081122105633/http://news.developer.nvidia.com/2008/02/cuda-11---now-o.html). February 14, 2008. Archived from the original (http://news.developer.nvidia.com/2008/02/cuda-11---now-o.html) on November 22, 2008.
16. Silberstein, Mark; Schuster, Assaf; Geiger, Dan; Patney, Anjul; Owens, John D. (2008). *Efficient computation of sum-products on GPUs through software-managed cache* (https://escholarship.org/content/qt8js4v3f7/qt8js4v3f7.pdf?t=ptt3te) (PDF). Proceedings of the 22nd annual international conference on Supercomputing – ICS '08. pp. 309–318. doi:10.1145/1375527.1375572 (https://doi.org/10.1145%2F1375527.1375572). ISBN 978-1-60558-158-3.
17. "CUDA C Programming Guide v8.0" (http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf) (PDF). *nVidia Developer Zone*. Section 3.1.5. January 2017. p. 19. Retrieved 22 March 2017.
18. "NVCC forces c++ compilation of .cu files" (https://devtalk.nvidia.com/default/topic/508479/cuda-programming-and-performance/nvcc-forces-c-compilation-of-cu-files/#entry1340190).
19. Whitehead, Nathan; Fit-Florea, Alex. "Precision & Performance: Floating Point and IEEE 754 Compliance for Nvidia GPUs" (https://developer.nvidia.com/sites/default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf) (PDF). Nvidia. Retrieved November 18, 2014.
20. "CUDA-Enabled Products" (https://www.nvidia.com/object/cuda_learn_products.html). *CUDA Zone*. Nvidia Corporation. Retrieved 2008-11-03.
21. "Coriander Project: Compile CUDA Codes To OpenCL, Run Everywhere" (http://www.phoronix.com/scan.php?page=news_item&px=CUDA-On-CL-Coriander). Phoronix.
22. Perkins, Hugh (2017). "cuda-on-cl" (http://www.iwocl.org/wp-content/uploads/iwocl2017-hugh-perkins-cuda-cl.pdf) (PDF). IWOCL. Retrieved August 8, 2017.

23. "hughperkins/coriander: Build NVIDIA® CUDA™ code for OpenCL™ 1.2 devices" (https://github.com/hughperkins/coriander). GitHub. May 6, 2019.
24. "CU2CL Documentation" (http://chrec.cs.vt.edu/cu2cl/documentation.php). *chrec.cs.vt.edu*.
25. "NVIDIA CUDA Programming Guide. Version 1.0" (http://developer.download.nvidia.com/compute/cuda/1.0/NVIDIA_CUDA_Programming_Guide_1.0.pdf) (PDF). June 23, 2007.
26. "NVIDIA CUDA Programming Guide. Version 2.1" (http://developer.download.nvidia.com/compute/cuda/2_1/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.1.pdf) (PDF). December 8, 2008.
27. "NVIDIA CUDA Programming Guide. Version 2.2" (http://developer.download.nvidia.com/compute/cuda/2_2/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.2.pdf) (PDF). April 2, 2009.
28. "NVIDIA CUDA Programming Guide. Version 2.2.1" (http://developer.download.nvidia.com/compute/cuda/2_21/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.2.1.pdf) (PDF). May 26, 2009.
29. "NVIDIA CUDA Programming Guide. Version 2.3.1" (http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf) (PDF). August 26, 2009.
30. "NVIDIA CUDA Programming Guide. Version 3.0" (http://developer.download.nvidia.com/compute/cuda/3_0/toolkit/docs/NVIDIA_CUDA_ProgrammingGuide.pdf) (PDF). February 20, 2010.
31. "NVIDIA CUDA C Programming Guide. Version 3.1.1" (http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf) (PDF). July 21, 2010.
32. "NVIDIA CUDA C Programming Guide. Version 3.2" (http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf) (PDF). November 9, 2010.
33. "CUDA 11.1 Release Notes" (https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html). *NVIDIA Developer*.
34. "CUDA 11 Features Revealed" (https://developer.nvidia.com/blog/cuda-11-features-revealed/). *NVIDIA Developer Blog*. 2020-05-14. Retrieved 2020-10-05.
35. "NVIDIA Quadro NVS 420 Specs" (https://www.techpowerup.com/gpu-specs/quadro-nvs-420.c1448). *TechPowerUp GPU Database*.
36. Larabel, Michael (March 29, 2017). "NVIDIA Rolls Out Tegra X2 GPU Support In Nouveau" (http://www.phoronix.com/scan.php?page=news_item&px=Tegra-X2-Nouveau-Support). Phoronix. Retrieved August 8, 2017.
37. Nvidia Xavier Specs (https://www.techpowerup.com/gpudb/3232/xavier) on TechPowerUp (preliminary)
38. "H.1. Features and Technical Specifications – Table 13. Feature Support per Compute Capability" (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-and-technical-specifications). *docs.nvidia.com*. Retrieved 2020-09-23.
39. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-and-technical-specifications
40. H.1. Features and Technical Specifications – Table 14. Technical Specifications per Compute Capability (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#features-and-technical-specifications)
41. ALUs perform only single-precision floating-point arithmetics. There is 1 double-precision floating-point unit.
42. Durant, Luke; Giroux, Olivier; Harris, Mark; Stam, Nick (May 10, 2017). "Inside Volta: The World's Most Advanced Data Center GPU" (https://devblogs.nvidia.com/inside-volta/). *Nvidia developer blog*.
43. No more than one scheduler can issue 2 instructions at once. The first scheduler is in charge of warps with odd IDs. The second scheduler is in charge of warps with even IDs.
44. "H.6.1. Architecture" (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#architecture-7-x). *docs.nvidia.com*. Retrieved 2019-05-13.
45. "I.7. Compute Capability 8.x" (https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capability-8-x). *docs.nvidia.com*. Retrieved 2020-09-23.
46. "Appendix F. Features and Technical Specifications" (http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf) (PDF). (3.2 MiB), page 148 of 175 (Version 5.0 October 2012).
47. "PyCUDA" (http://mathema.tician.de/software/pycuda).
48. "pycublas" (https://web.archive.org/web/20090420124748/http://kered.org/blog/2009-04-13/easy-python-numpy-cuda-cublas/). Archived from the original (http://kered.org/blog/2009-04-13/easy-python-numpy-cuda-cublas/) on 2009-04-20. Retrieved 2017-08-08.
49. "CuPy" (https://cupy.chainer.org/). Retrieved 2020-01-08.
50. "nVidia CUDA Bioinformatics: BarraCUDA" (https://www.biocentric.nl/biocentric/nvidia-cuda-bioinformatics-barracuda/). *BioCentric*. 2019-07-19. Retrieved 2019-10-15.
51. "Part V: Physics Simulation" (https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation). *NVIDIA Developer*. Retrieved 2020-09-11.

External links

- Official website (https://developer.nvidia.com/cuda-zone)