At its core are three key abstractions - a hierarchy of thread groups, shared memories, and barrier synchronization - that are simply exposed to the programmer as a minimal set of language extensions.

There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads.

* Private Memory for each thread
* Shared Memory for each block
* Global Memory can be accessed by all threads
* ReadOnlyMemory – Constants and Textures accessible by all threads.
* In Cuda, DRAM = > host memory and device memory

Unified Memory provides managed memory to bridge the host and device memory spaces. Managed memory is accessible from all CPUs and GPUs in the system as a single, coherent memory image with a common address space. This capability enables oversubscription of device memory and can greatly simplify the task of porting applications by eliminating the need to explicitly mirror data on host and device.

The compute capability of a GPU and the features supported are represented by a version number/"SM version". But Cuda software version is another.

Any source file that contains some of these extensions must be compiled with NVCC.

Kernels operate out of device memory, so the runtime provides functions to Programming Interface to allocate, deallocate, and copy device memory, as well as transfer data between host memory and device memory.

Device memory can be allocated either as linear memory or as CUDA arrays. Linear memory exists on the device in a 40-bit address space, so separately allocated entities can reference one another via pointers, for example, in a binary tree.

