Pune Vidyarthi Griha's

COLLEGE OF ENGINEERING, NASHIK – 3.

"CUDA ARCHITECTURE"

By

Prof. Anand N. Gharu

(Assistant Professor)

PVGCOE Computer Dept.

20th July 2018

Topic Overview

- CUDA Architecture
- Using the CUDA Architecture
- Applications of CUDA
- Introduction to CUDA C-Write and launch CUDA C kernels
- Manage GPU memory
- Manage communication and synchronization
- Parallel programming in CUDA- C.

INTRODUCTION OF CUDA

- CUDA is a set of developing tools to create applications that will perform execution on GPU (Graphics Processing Unit).
- CUDA compiler uses variation of C with future support of C++.
- CUDA was developed by NVidia and can only run on NVidia GPUs of tesla and Geforce series.
- CUDA provides Heterogeneous serial-parallel computing Between CPU and GPU
- CUDA is a platform for performing massively parallel computations on graphics accelerators.
- CUDA was developed by NVIDIA
- It was first available with their G8X line of graphics cards
- CUDA is supported on all of NVIDIA’s G8X and above graphics cards
- The current CUDA GPU Architecture is branded Tesla

INTRODUCTION OF CUDA

- CUDA provides ability to use high-level languages such as C to develop application that can take advantage of high level performance and scalability that GPUs architecture offer.
- GPUs allow creation of very large number of concurrently executed threads at very low system resource cost.
- CUDA also exposes fast shared memory (16KB) that can be shared between threads.
- Full support for integer and bitwise operations.
- Compiled code will run directly on GPU.
- CUDA is a parallel computing platform and programming model developed by Nvidia for general computing on its own GPUs (graphics processing units). CUDA enables developers to speed up compute-intensive applications by harnessing the power of GPUs for the parallelizable part of the computation

INTRODUCTION OF GPU

- A Graphics Processing Unit (GPU) is a microprocessor that has been designed specifically for the processing of 3D graphics.
- The processor is built with integrated transform, lighting, triangle setup/clipping, and rendering engines, capable of handling millions of **math-intensive processes** per second.
- GPUs form the heart of modern graphics cards, relieving the CPU (central processing units) of much of the graphics processing load.
- GPUs allow products such as desktop PCs, portable computers, and game consoles to process real-time 3D graphics that only a few years ago were only available on high-end workstations.
- Used primarily for 3-D applications, a graphics processing unit is a single-chip processor that creates lighting effects and transforms objects every time a 3D scene is redrawn.
- These are **mathematically-intensive tasks,** which otherwise, would put quite a strain on the CPU. Lifting this burden from the CPU frees up cycles that can be used for other jobs.

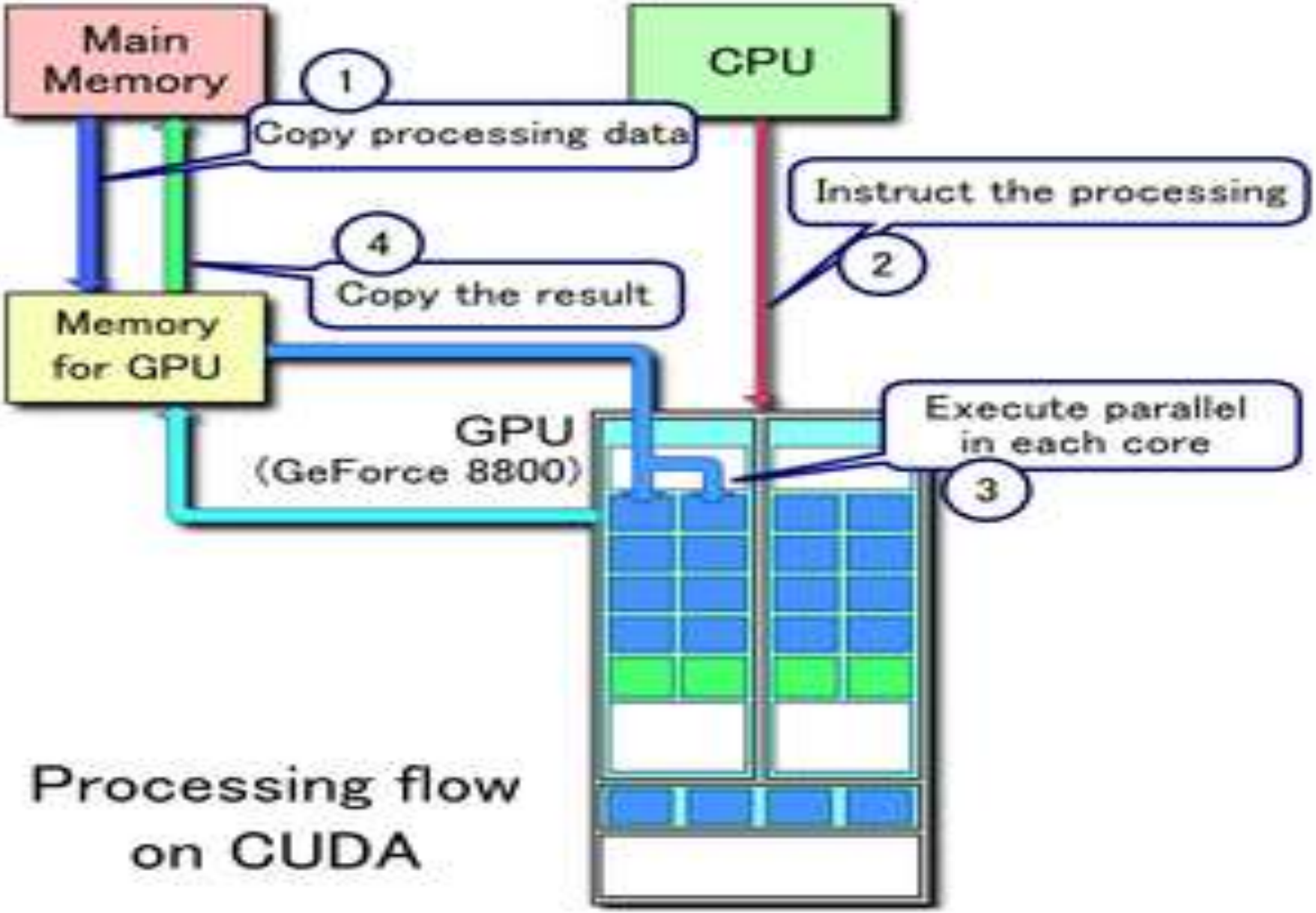
CUDA ARCHITECTURE

- CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model created by Nvidia.
- It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing.
- 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.
- The CUDA platform is designed to work with programming languages such as C, C++, and Fortran.

Flow of Cuda Archirecture :

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

CUDA ARCHITECTURE (FLOW OF CUDA)



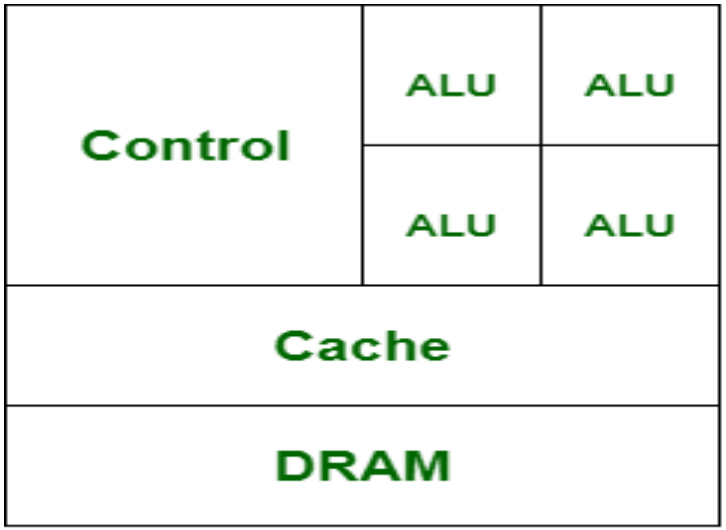
CUDA ARCHITECTURE

- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel

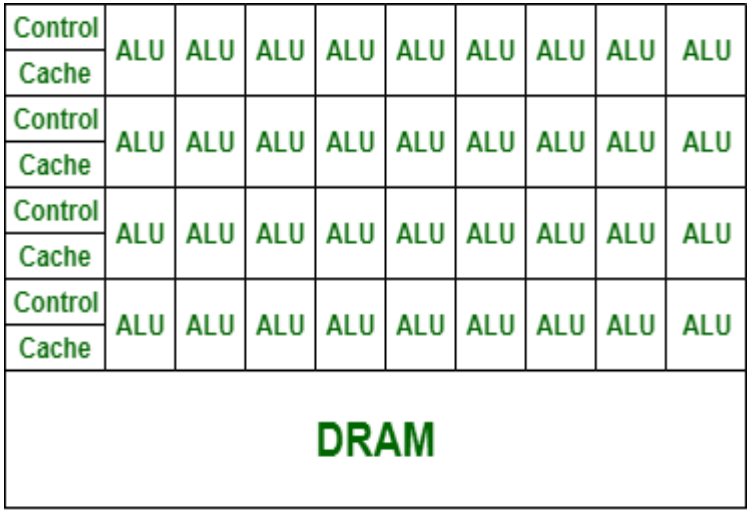
- Data-parallel portions of an application are executed on the device as kernels which run in parallel on many threads

- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
  - Very little creation overhead
  - GPU needs 1000s of threads for full efficiency
  - Multi-core CPU needs only a few

CPU VS GPU



CPU



GPU

- Less than 20 cores
- 1–2threads per core
- Latency is hidden by large cache

More than 512 cores
10s to 100s of threads per core
Latency is hidden by fast context switching

GPUs don’t run without CPUs

CPU VS GPU

S.NO	CPU	GPU
1.	CPU stands for Central Processing Unit.	While GPU stands for Graphics Processing Unit.
2.	CPU consumes or needs more memory than GPU.	While it consumes or requires less memory than CPU.
3.	The speed of CPU is less than GPU's speed.	While GPU is faster than CPU's speed.
4.	CPU contain minute powerful cores.	While it contain more weak cores.
5.	CPU is suitable for serial instruction processing.	While GPU is not suitable for serial instruction processing.
6.	CPU is not suitable for parallel instruction processing.	While GPU is suitable for parallel instruction processing.
7.	CPU emphasis on low latency.	While GPU emphasis on high throughput.

APPLICATIONS CUDA

1. Fast Video Transcoding

Transcoding is a very common, and highly complex procedure which easily involves trillions of parallel computations, many of which are floating point operations. Applications such as Badaboom have been created which harness the raw computing power of GPUs in order to transcode video much faster than ever before. For example, if you want to transcode a DVD so it will play on your iPod, it may take several hours to fully transcode. However, with **Badaboom**, it is possible to transcode the movie or any video file faster than real time. **(e.g. AVC – any video converter)**

2. Medical Imaging

CUDA is a significant advancement for the field of medical imaging. Using CUDA, MRI machines can now compute images faster than ever possible before, and for a lower price. Before CUDA, it used to take an entire day to make a diagnosis of cancer or any other disease. Now with CUDA, this can take 30 minutes. In fact, patients no longer need to wait 24 hours for the results, which will benefit many people**.**

APPLICATIONS CUDA

3. Oil and Natural Resource Exploration

The first two topics I talked about had to do with video, which is naturally suited for the video card. Now it’s time to talk about more serious technologies involving oil, gas, and other natural resource exploration. Using a variety of techniques, it is overwhelmingly difficult to construct a 3d view of what lies underground, expecially when the ground is deeply submerged in a sea. Scientists used to work with very small sample sets, and low resolutions in order to find possible sourses of oil. Because the ground reconstruction algorithms are highly parallel, CUDA is perfectly suited to this type of challenge. Now CUDA is being used to find oil sources quicker.

4. Computational Sciences

In the raw field of computational sciences, CUDA is very advantageous. For example, it is now possible to use CUDA with MATLAB, which can increase computations by a great amount. Other common tasks such as computing eigenvalues, or SVD decompositions, or other matrix mathematics can use CUDA in order to speed up calculations.

APPLICATIONS CUDA

5. Neural Networks

they personally worked on a program which required the training of several thousand neural networks to a large set of training data. Using the Core 2 Duo CPU that was available to them, it would have taken over a month to get a solution. However, with CUDA, they were able to reduce their time to solution to under 12 hours.

6. Gate-level VLSI Simulation

it is used simulate VLSI circuit into modelling to appear on the screen. It is easy to understand the concept of internal circuit.

.

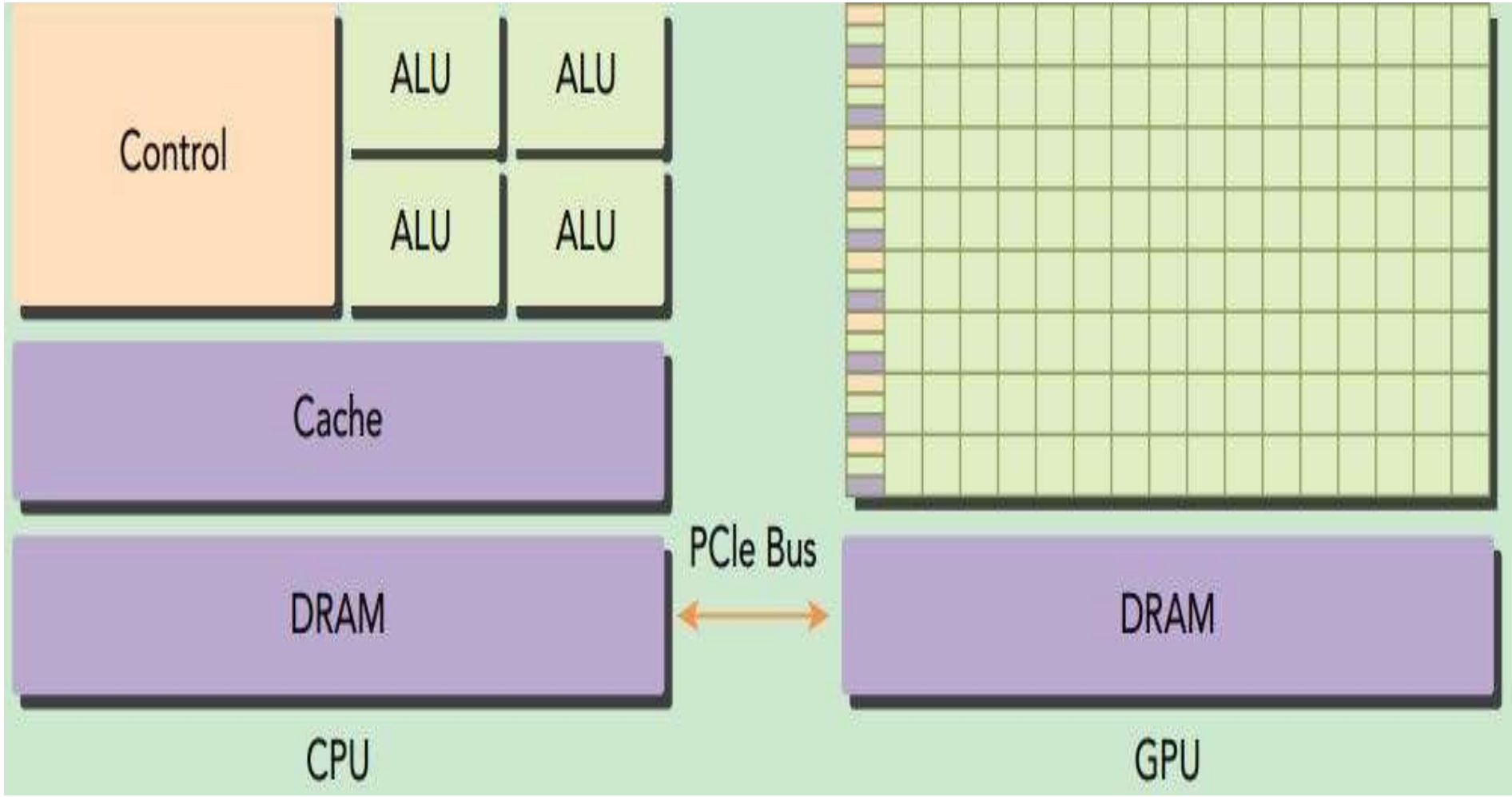
7. Fluid Dynamics

Fluid dynamics simulations have also been created. These simulations require a huge number of calculations, and are useful for wing design, and other engineering tasks.

Hetrogeneous Architecture in CUDA

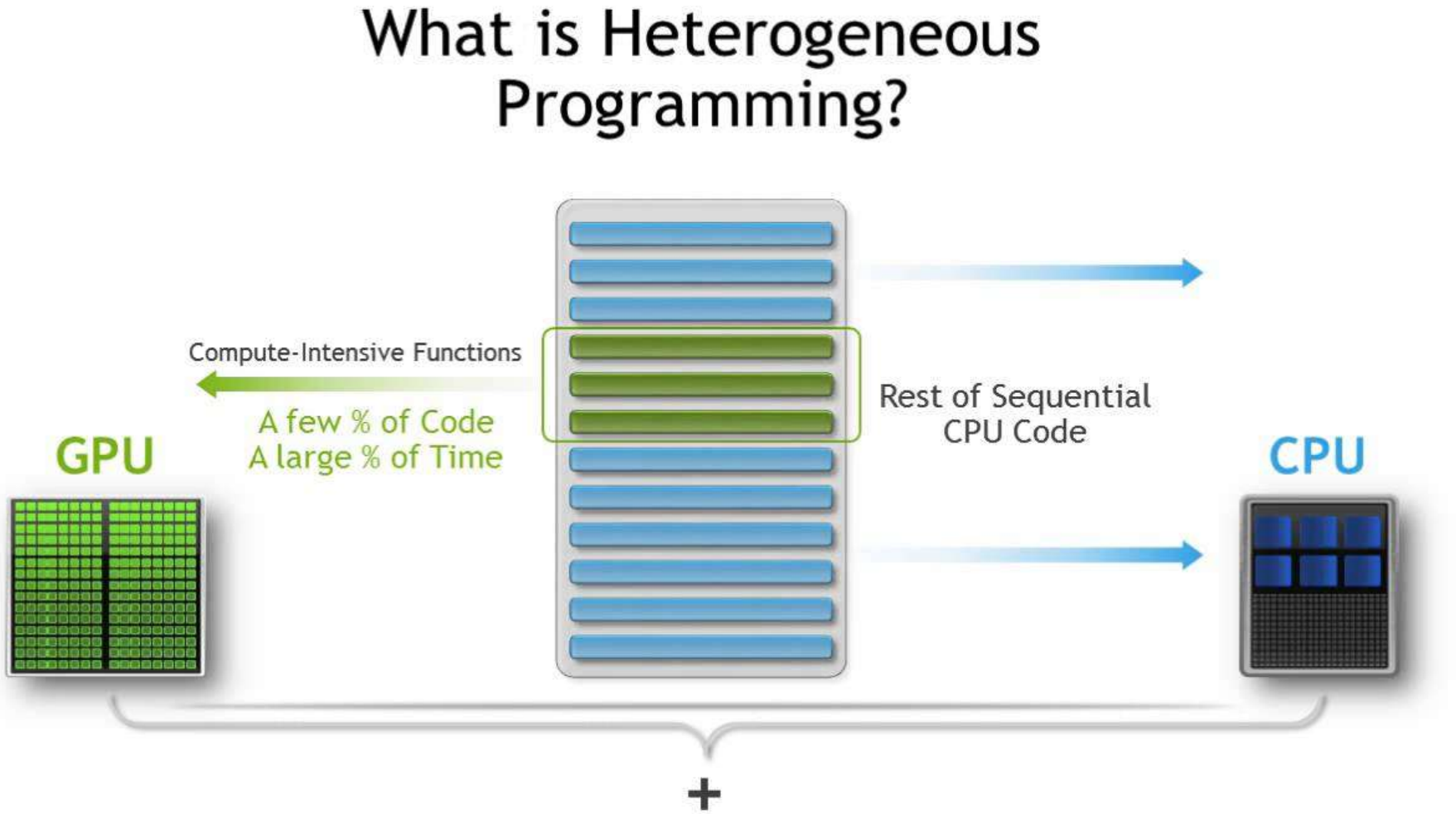
- Heterogeneous System Architecture (HSA) is a cross-vendor set of specifications that allow for the integration of central processing units and graphics processors on the same bus, with shared memory and tasks.
- The HSA is being developed by the HSA Foundation, which includes (among many others) AMD and ARM.
- The platform's stated aim is to reduce communication latency between CPUs, GPUs and other compute devices.
- CUDA and OpenCL as well as most other fairly advanced programming languages can use HSA to increase their execution performance.
- Heterogeneous computing is widely used in system-on-chip devices such as tablets, smartphones, other mobile devices, and video game consoles.
- HSA allows programs to use the graphics processor for floating point calculations without separate memory or scheduling.

Hetrogeneous Architecture in CUDA

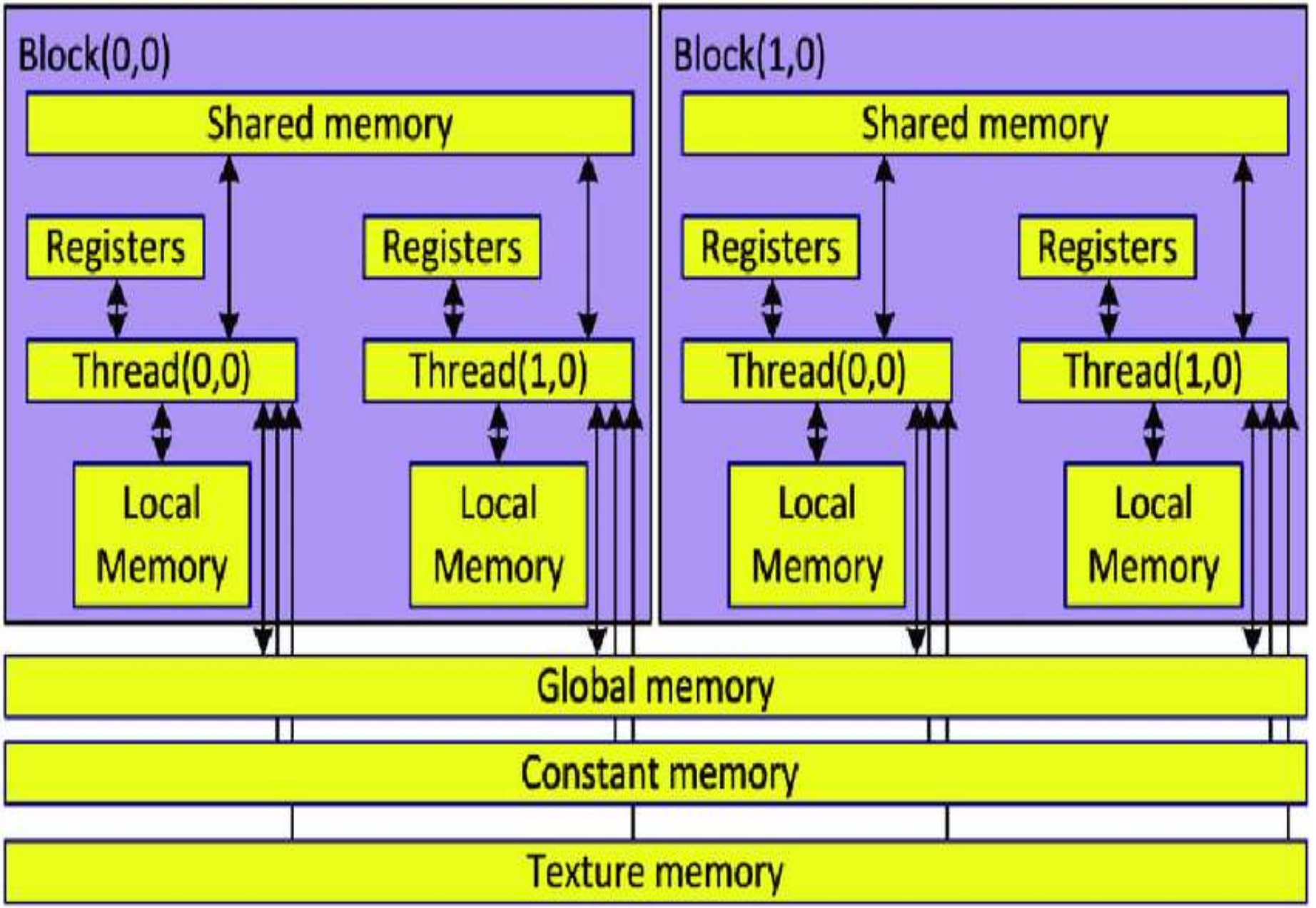


Hetrogeneous Architecture in CUDA

Heterogeneous computing refers to systems that use more than one kind of processor or cores. These systems gain performance or energy efficiency not just by adding the same type of processors, but by adding dissimilar coprocessors, usually incorporating specialized processing capabilities to handle particular tasks.



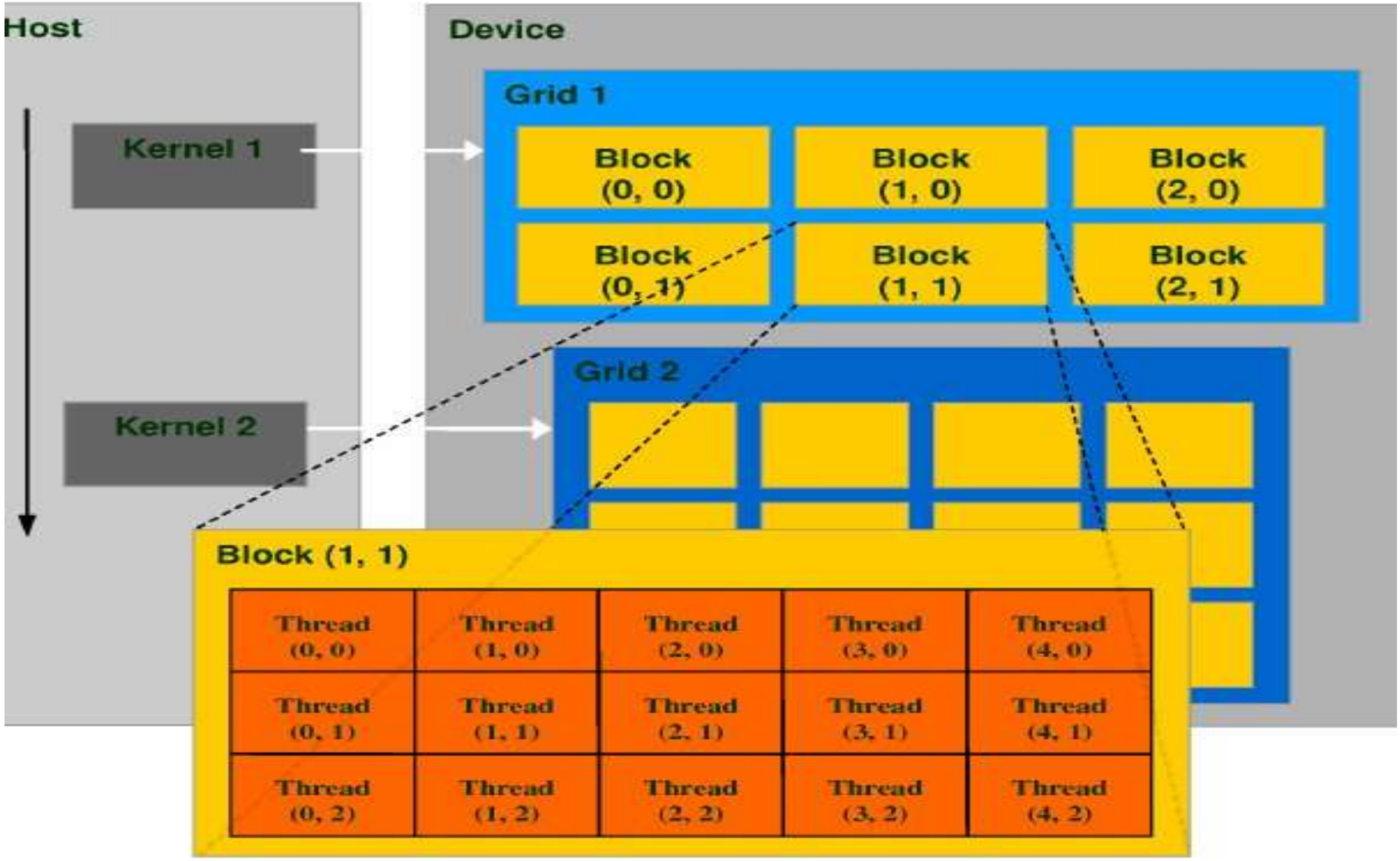
MEMORY ORGANIZATION IN CUDA



MEMORY ORGANIZATION IN CUDA

- CUDA uses segmented memory architecture that allow appli to access data in global, local, shared, constant and texture memory.
- There are several level of memory & memory org are arranged in the hierarchical fashion and each leve has distinct Read and write characteristics.
- Every primitive thread has access to private "local" memory as well as registers. Every thread in thread block has access to unified 'shared memory', shared among all threads for the life of that thread block. Finally all thread have read/write accen to global memory.
- There also exist a read-only 'constant' & texture memory in the same loc at global memory.
- Global memory is not cached, thro. mem. transactions may be 'coalesced' to hide the high memory access latency.
- The read only constant memory resides in the same location as global memory. this mem. may be cached.
- The read only texture mem. resides in same loc of global mem. and is also cached.

THREAD ORGANIZATIONN CUDA



THREAD ORGANIZATION CUDA

- The CUDA processing paradigm uses an approach is called kernel. A kernel is actualy a sub-routine or mini-prog.
- The device like NVIDIA graphics card used inside the host sys. runs kernel.

The kernel prog. is executed simultaneously by nos of primitives thread.

- There are nos of batches of primitives threads organized into thread blocks. A thread block contains a specific nos of primitives threads, chosen based on the amount of available shared memory as well as mem. latency hiding tech characteristics desired.

THREAD ORGANIZATION CUDA

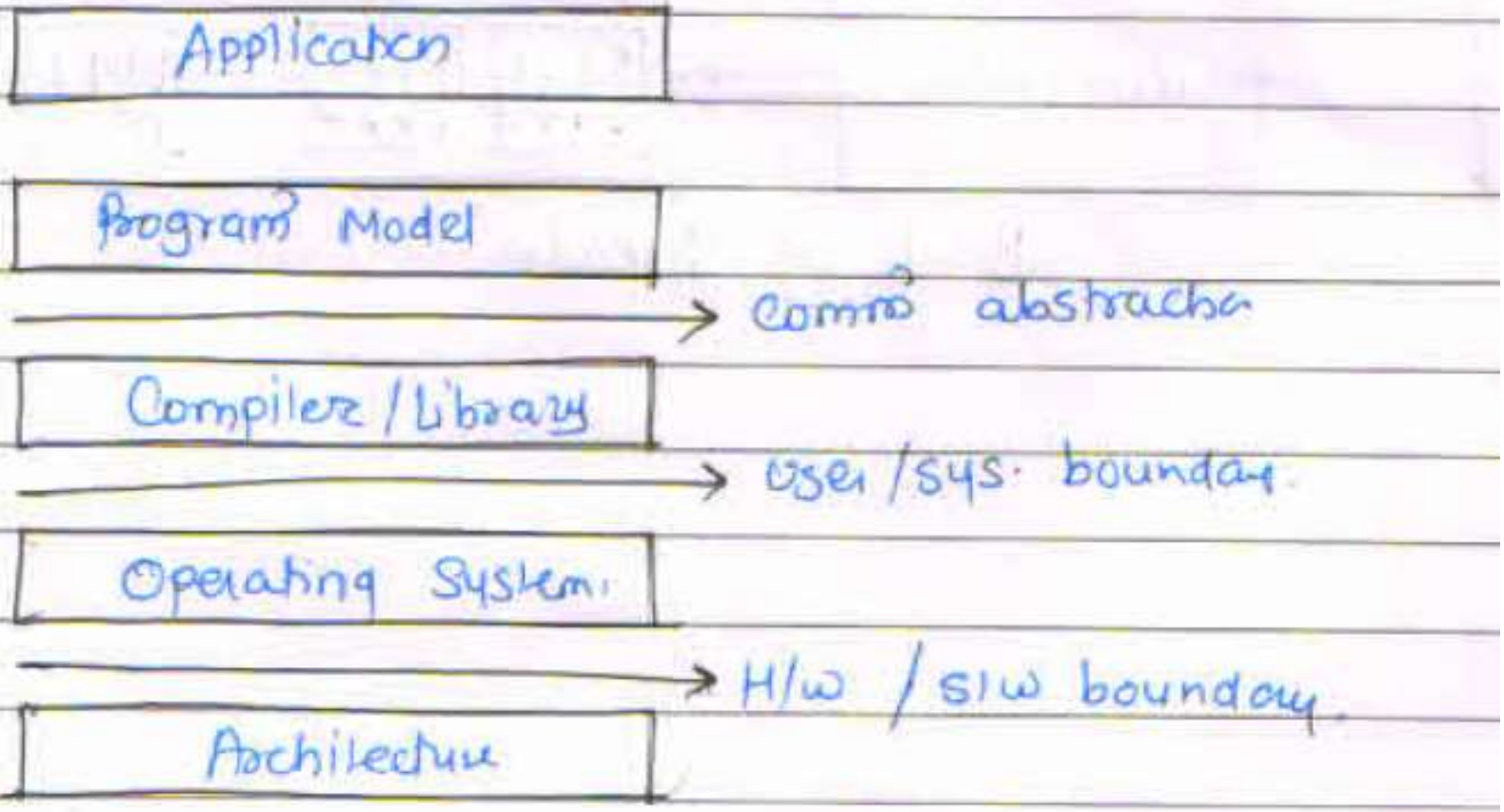
The nos g thread in thread block is also limited by archi. to total g 512 thread per block. Such thread within thread block can Communicate efficiently using shared memory scoped to each thread block.

Using this shared mem, all thread can also synd. within thread block.

- Every thread within thread block has its own thread ID.
- Thread blocks are organized into 1D, 2D or 3D array.
- As shown in fig. a grid which is collection g thread block g the same thread which all executes the same kernel. The thread block are physically limited to 512 threads per block. & bcz g that grids are used for Computing a large nos g thread block in kel.
- Thread block in grid may not synchronize with one another bcz they may not communicate via shared memory
- The diag shows the thread hierarchy. In this represe. a given kernel contain 3X2 grid g thread block.
- There are total 72 threads executing in the said kernel where each thread block is a 4X3.

CUDA PROGRAMMING MODEL

- The model used for prg in parallel archite. acts as a bridge bet an appli and its implem on available h/w.
- Fig shows layers of abstraction lies bet the prog and prg model implementation.

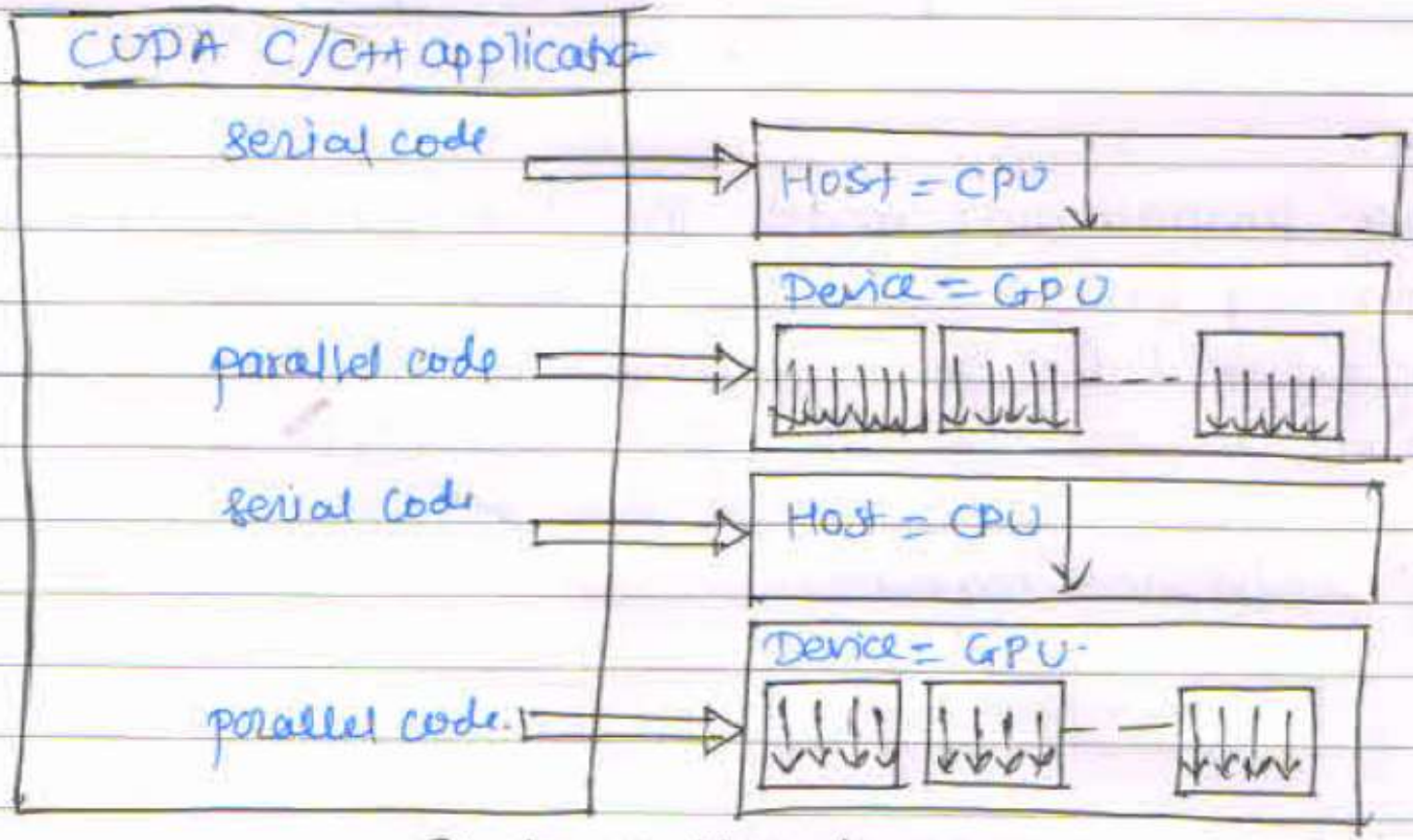


CUDA PROGRAMMING MODEL

- The boundary bet prog & prog model impln is comm abstraction. A compiler / libraries using sys. calls to access hw services. are used to provide such a comm abstraction.
- The components of the prog share infor and coordinates their activities by the instns provided thro prog.
- The CUDA prg model share many abstraction with other parallel prg models. it has some features of fully utilization of GPU.

- Anatomy of CUDA C/C++ Application:

- The appli for the archite. contains serial and ||el code shown in fig.
- The serial code executes in a host (CPU) threads whereas the ||el code executes in many devices (GPU) threads across multiple processing elements.



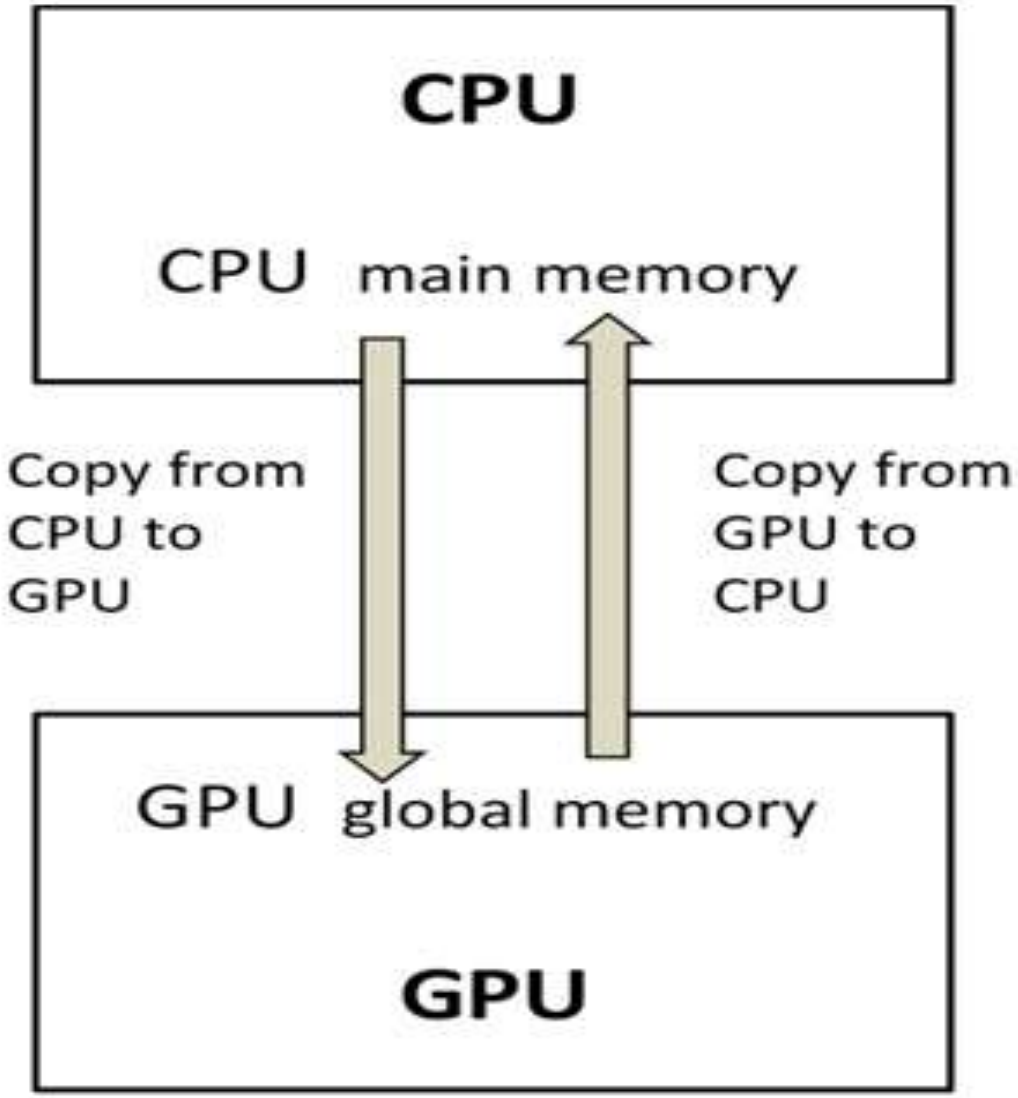
NVIDIA TESLA GPU

- GPU (graphics Processing Unit) is basically used for 3-D applicahcm. A GPU is single chip procesor used to provide lighting effects. and transforms objects every time a 3-D scene is redraion..
- The GPU is used to assist CPU on mathemabcally - intesive tasks. require in performing graphics related achvities. The CPU is freed from the overheads occurred from graphics related mathemabc - intensive tasks and the free cycles of CPU. can be used for other tasks.
- The GPU is used along with the CPU to accelerate scienbfic, analytics, engineering etc. appli's. The overall computing is accelerated by the GPU and therefore this filed of computing is called GPU accelerated Computing.

NVIDIA TESLA GPU

- The basic difference bet' a CPU & GPU is the use g nos g cores included. The CPU contains few nos g cores for sequential serial procesing whereas a GPU contains more than thousands g smaller and efficients cores for handling multiple tasks simultaneously.
- Nvidia's brand name for the product, they develop for stream procesing and general purpose GPUs is called as Nvidia Tesla.
- The GPUs from G80 series onwards are utilized in the Nvidia Tesla Products.

GPU PROGRAMMING MODEL



GPU PROGRAMMING MODEL

- As we know the distinction bet' Host and device used in CUDA prog' model.
- The code executed in host side is the part g code executed on the CPU. & this will include RAM and harddisk.
- whereas, code executed on device is automatically loaded on the graphic card and run on latter.
- Another imp concept is kernel; it stands for fn' ~~prog~~. performed on the device and launched from the host.
- The code defined in the Kernel will be performed in ||el. by an array g thread
- Below fig. shows how GPU prog' model works.

GPU PROGRAMMING MODEL

- The running prog will have source code to run on GPU. and code to run on GPU.
- CPU & GPU have seperated memories.
- The data is transfered from CPU to GPU to be computed.
- The data o/p from GPU Computation is copied back to. CPU memory.

INTRODUCTION TO CUDA C

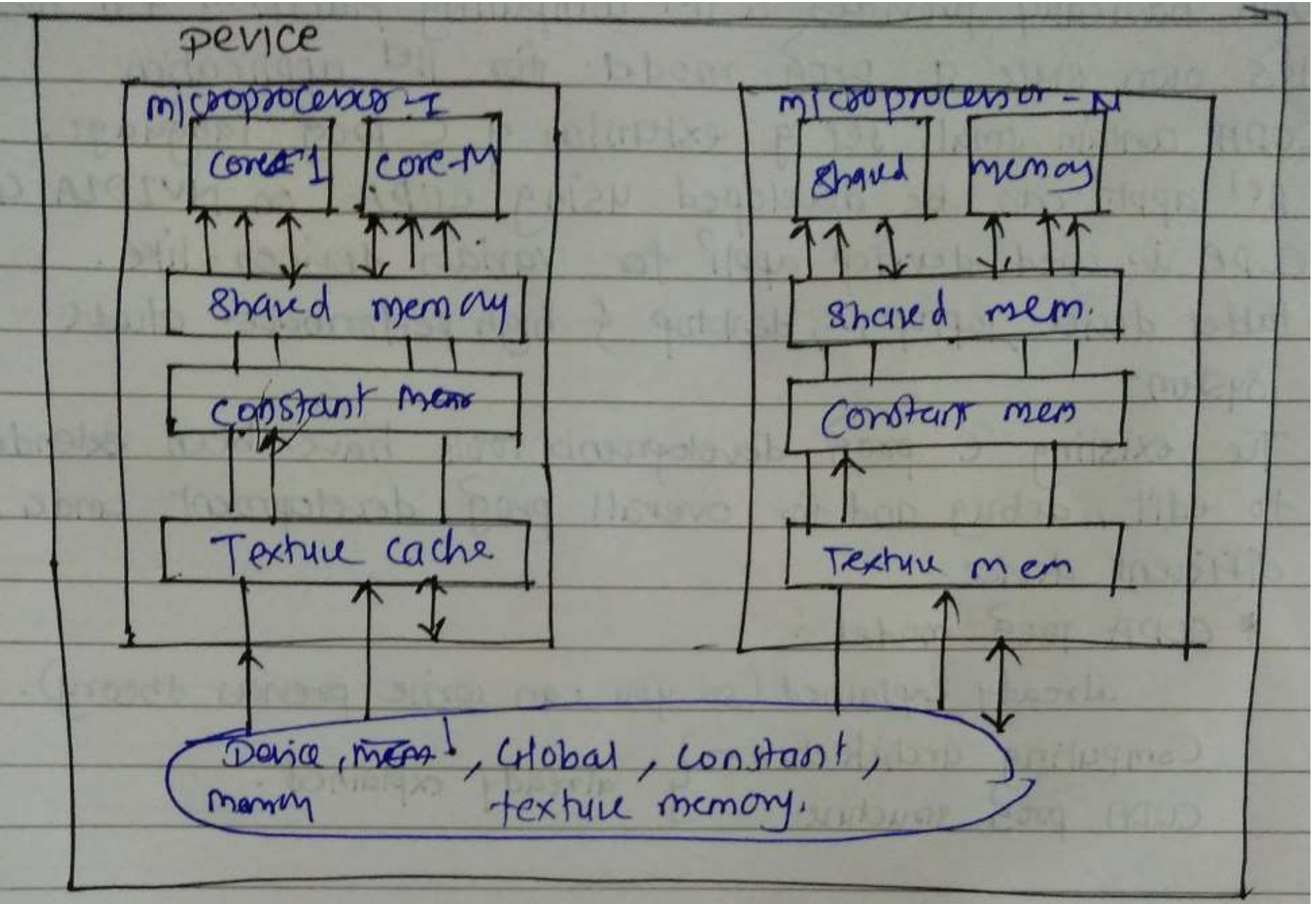
- CUDA basically provides a ||el computing platform & it has it's own style & prog model for ||el application
- CUDA contain small set of extension of C prog language. ||el appl? can be developed using CUDA on NVIDIA GPU
- CUDA is used develop appl? for various devices like tablet device, laptops, desktop & high performance clust system.
- The existing 'C' prog development tools have been extended to edit, debug and for overall prog development unde different device.

* CUDA prog model :-

already explained (so you can write previas theory).

Computing architecture :-
CUDA prog structure :-
} already explained.

HOW TO MANAGE GPU MEMORY



HOW TO MANAGE GPU MEMORY

- large global device memory is available. & accessible by microprocessor for both gather & scatter operation.
- This mem. does not provide. cacheing & hence it is slow.
- As compared to device shared memory is fast. & it takes same amount of time. as required to access register.
- Shared mem. is also called as parallel data cache (PDC) & is local to each microprocessor. unlike device memory.
- Shared mem. is divided into many part for synchronization.
- Shared mem. access is restricted to block.
- Each thread in multiprocessor has it's own part of shared memory.
- Shared memory space is limited and should be used efficiently.

ADVANTAGES OF CUDA

1. Programming interface of CUDA applications is based on the standard C language with extensions, which facilitates the learning curve of CUDA
2. CUDA provides access to 16 KB of memory (per multiprocessor) shared between threads, which can be used to setup cache with higher bandwidth than texture lookups
3. More efficient data transfers between system and video memory
4. No need in graphics APIs with their redundancy and overheads
5. Linear memory addressing, gather and scatter, writing to arbitrary addresses
6. Hardware support for integer and bit operations

LIMITATIONS OF CUDA

1. No recursive functions
2. Minimum unit block of 32 threads
3. Bus bandwidth and latency between CPU & GPU may be bottleneck.
4. Only supported on NVIDIA GPU’s
5. Closed CUDA architecture, it belongs to NVIDIA.

Synchronization between Threads

- The CUDA API has a method, **__syncthreads()** to synchronize threads. When the method is encountered in the kernel, all threads in a block will be blocked at the calling location until each of them reaches the location.

- What is the need for it? It ensure phase synchronization. That is, all the threads of a block will now start executing their next phase only after they have finished the previous one.

- For example, if a __syncthreads statement, is present in the kernel, it must be executed by all threads of a block.

- If it is present inside an if statement, then either all the threads in the block go through the if statement, or none of them does.

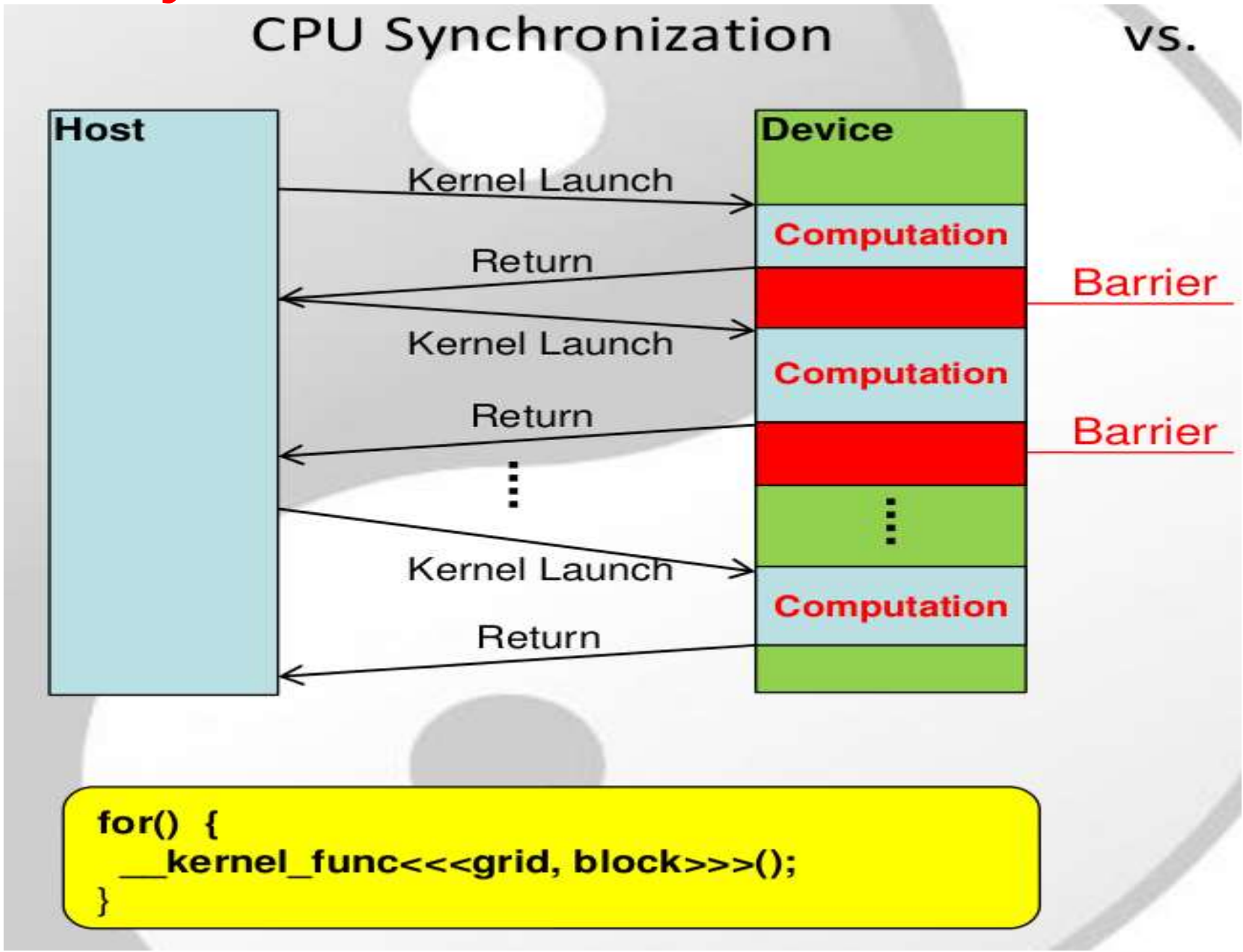
Synchronization between thread

- If an if-then-else statement is present inside the kernel, then either all the threads will take the **if** path, or all the threads will take the else path.

- This is implied. As all the threads of a block have to execute the sync method call, if threads took different paths, then they will be blocked forever.

- It is the duty of the programmer to be wary of such conditions that may arise.

Synchronization between thread



Applications

Applications

- 3D image analysis
- Adaptive radiation therapy
- Astronomy
- Automobile vision
- Bio informatics
- Biological simulation
- Broadcast
- Computational Fluid Dynamics
- Computer Vision
- Cryptography
- CT reconstruction
- Data Mining
- Electromagnetic simulation
- Equity training
- Financial - lots of areas
- Mathematics research
- Military (lots)
- Mine planning
- Molecular dynamics
- MRI reconstruction
- Network processing
- Neural network
- Protein folding
- Quantum chemistry
- Ray tracing
- Radar
- Reservoir simulation
- Robotic vision/AI
- Robotic surgery
- Satellite data analysis
- Seismic imaging
- Surgery simulation

THANK YOU !!!!

My Blog : https://anandgharu.wordpress.com/

Email : gharu.anand@gmail.com