U-5 (HPC)

Page No.
Date

* Intro to GPU-

- age of parallel computing → multi tasking phones, tablets, laptops, etc.
- Provide novel, fast & rich - experience to user, developer need to understand and & use various parallel platform.
- CPU performance increasing no. of cores transistor & other features & always has been a ~~relatatate~~ reliable way of enhancing performance but only way.

• GPU's -
  • specialised electronic circuit. Helps alter memory rapidly to acc img, given as output to display device.
  • GPUs are in - Mobile Phones [Adreno/ Mali], PCs (AMD / NVIDIA) consoles [PS4/5, XBOX].
  • GPUs are great at img processing & controlling computer graphics.
  • They are highly parallel.

• CUDA - • Compute Unified Device Architechure
  • launched by NVIDIA in 2007 as an programming ~~language~~ interface providing parallel computation using GPUs.

•

• CUDA architecture -
  • Before CUDA archi. vertex & pixel shaders were used for parallel computing.
  • pixel shader is a GPU component that can operate per pixel.
  • vertex shader is also - a GPU component like pixel & it's assembly language specific, used for geometrical operations.
  • · an extension of C++ features to support GPU acceleration.

components → • parallel compute engines inside NVIDIA GPUs
  • OS kernel level support for hardware.
  • Usermode driver, • device level API for developers
  • PTX instruction set architecture for parallel computing

- Bioinformatics.
- Fast Video Transcoding

Page No.
Date

- Appl^n of CUDA ⟶ • Deep Learning • Computing • Computer Vision.
- CUDA → general purpose computing • Data Science • Ultra sound imaging
- 3 Device Level APIs → 1) OpenCL 2) Direct X 3) CUDA Driver API.
- Language Level - 3) i) Fortan ii) C++ iii) C.

* Processing flow of Cuda-C.

1) Host Code Excecution:
2) Kernel Launch
3) Kernel Excecution on GPU:
4) Data Access & Processing
5) Synchronization (optional).
6) Results Transfer (optional).
7) CleanUp -

* CUDA Kernel program -

```
#include <iostream>
__global__ void kernel (void) {
}
int main () {
    kernel <<< 1, 1>>> ();
    printf ("Hello World! \n");
    return 0;
}
```

* CUDA add^n of two vectors

```
cuda
__global__ void vectorAdd
(float *a, float *b, float *c, int n)
{ int i = blockIdx.x * blockDim.x
   + threadIdx.x;
  if (i<n)
  c[i] = a[i] + b[i];
}
```

- global keyword → helps call kernel function using host,
- global helps run the kernel fun^n on GPU.
- Use NVCC compiler to complet this code instead of gcc.
- kernel <<< 1, 1 >>> () is a CUDA specific syntax specifies call to device code.
- specifies a call to device code.
- parameters inside "<<< >>>" are called execution configuration.

* **Device -**
  - Refers to GPU (Graphics Processing Unit) in CUDA programming - executes device code & performs parallel computations on data.
  - Managed by CUDA runtime system for task scheduling & memory mgm.

* **Host -**
  - represents Central Processing Unit) in CUDA programming.
  - Executes host code responsible for managing device operations, memory
  - Data transfer & control execution flow of CUDA programs & interacts with GPU.

* **Device Code -**
  - Code written specifically to be executed on GPU in CUDA programy
  - Utilizes CUDA libraries & functions for parallel processing tasks.
  - can be invoked parallel processing tasks from host code to perform computations on GPU cores.

* **Kernel -**
  - function in CUDA programming designed to be executed in parallel GPUs
  - functions are identified by name & configuration, no of threads per block & blocks per grid.
  - Each thread executes kernel code independently processing diff portions of data simultaneously to active parallelism.

* **warps** - set of 32 concurrent threads in a block.

Page No.
Date

* Global Memory -

- Largest memory space available to CUDA programs.
- Resides on device & is typically used to store data needs to multiple threads.

* Shared Memory -

- shared memory is fast, on-chip memory space shared by all threads within single block.
- Much faster than global memory but limited capacity.
- Shared memory is used for data that needs to be accessed frequently & efficiently within a block.

* Constant Memory -

- Is also located on device & is read only for all threads. It
- It is cached & provides fast access to data accessed uniformly by all threads within a block.

* Thread Hierarchy

1) Grids → • grid is highest level of organizations in CUDA & multiple blocks.
   • represents overall computational workload & is executed on GPU device.
2) Blocks → • subdivided into blocks, each containing threads.
   • executed independently & scheduled on any multiprocessor on GPU.
3) Threads → • smallest unit of execution in CUDA & organized into blocks.
   • threads within a block can cooperate & synchronize with each other using shared memory & barriers.

- Block dimensions - • refers to no. of no. of threads per block.
  - specified as a 3-dimensional array of integers.
  - each block can contain up to a max. no. of threads (depend on GPU).
  - Dimension is crucial for organizing threads.

Page No.
Date

* Grid Dimension -

- Refers to no. of blocks in grid.
- specified as a 3d array of integers.
- grid dimensions determines overall size of computation & how blocks are organized for execution on GPU.
- Mutiple blocks withing grid can execute concurrently on diff streaming multiprocessors (SMs) on GPU.

CUDA kernel for adding two vectors element-wise-

```
__global__ void vectorAddition(float a , flot b, float
result, int size){
int index = blockIdx.x * blockDim.x + threadIdx.x;
    if (index < size) {
        result [index] = a[index] + b[index];
    }
}
```

* Kernel ~~Execution~~ in CUDA -

- CUDA → kernel is a function that is executed on GPU.
- written using CUDA C/C++ syntax & is responsible for performing parallel platforms computations.
- Kerness are launched from CPU & executed by mutiple threads on GPU.
- executes same kernel code with diff data, enabling massive parallelism.
- invoke CUDA kernel from CPU.
- kernel is launched, CUDA runtime system, manages allocation of resources on GPU & schedules execution of threads.
- CUDA kernel, while GPU executes kunel in parallel.

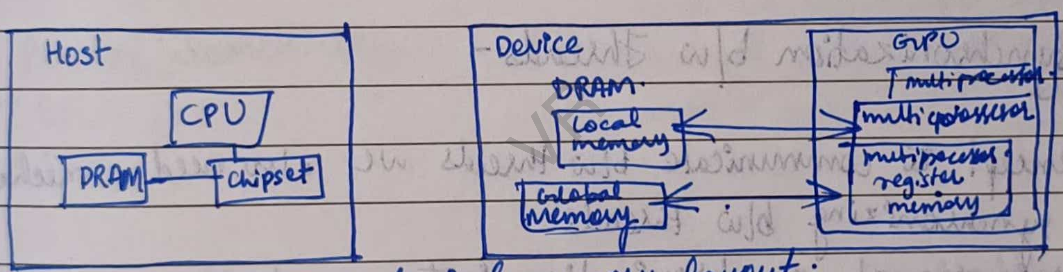
Page No.
Date

* CUDA kernel & Handling errors :-

- CUDA as extension of C consists of host code (program control) & device code (GPU) combined in a single C Program.
- CUDA is C for parallel processor.
- you can write a program for one thread & instantiate it on many parallel threads.

* CUDA memory model / GPU memory.

① CPU & GPU have seprate memory spaces.
② Device pointers point to GPU memory.
③ Host pointers point to GPU memory.
④ Host CPU code Manages device GPU Memory.



physical memory layout.

⑤ does following tasks -
  i) Allocate or free money.
  ii) copy data to & from device.
  iii) Applies to global device memory (DRAM).
⑥ Local memory resides in device DRAM.
⑦ Host can read & write global memory but not shared memory).
⑧ CUDA threads may access data from mutiple memory spaces during execution.
⑨ Each thread has private local memory.
⑩ All threads have acess to same global memory.

Page No.
Date

* Manage communication betⁿ threads -

- possible & many times necessary for threads within same block to communicate with each other.
- Need to share data b/w threads to compute final resnt or outputs them
- commnication b/w threads → possible using shared memory.
- block of threads shares memory called shared memory.
- variables in shared memory treated different than typical variable varialded by CUDA C complies.
- creates a copy of variable for each block that you launch on GPU
- threatt with same block communicate data with each other using shared memory or global memory.

* Synchronization b/w Threads -

- except to communicate b/w threeds we also need a mechanism for synchronizing b/w threads.
- Main use of synchronization is to prevent RAW, WAR, WAW hazards during communication b/w mutiple threads.
  - RAW (Read After Write)
  - WAR (Write After Read)
  - WAW (Writer After Write).

synchronize to commit all memory writers, reads & ...
③ CUDA synchronizes threads using function - syncthreads().
④ there are two types of synchronization in CUDA.
  1) Implicit Synchronization. 2) Explicit Synchronization.
⑤ Barrier a programmer can plane the synchronization barrier explicity to synchronize tasks.