Unit V.

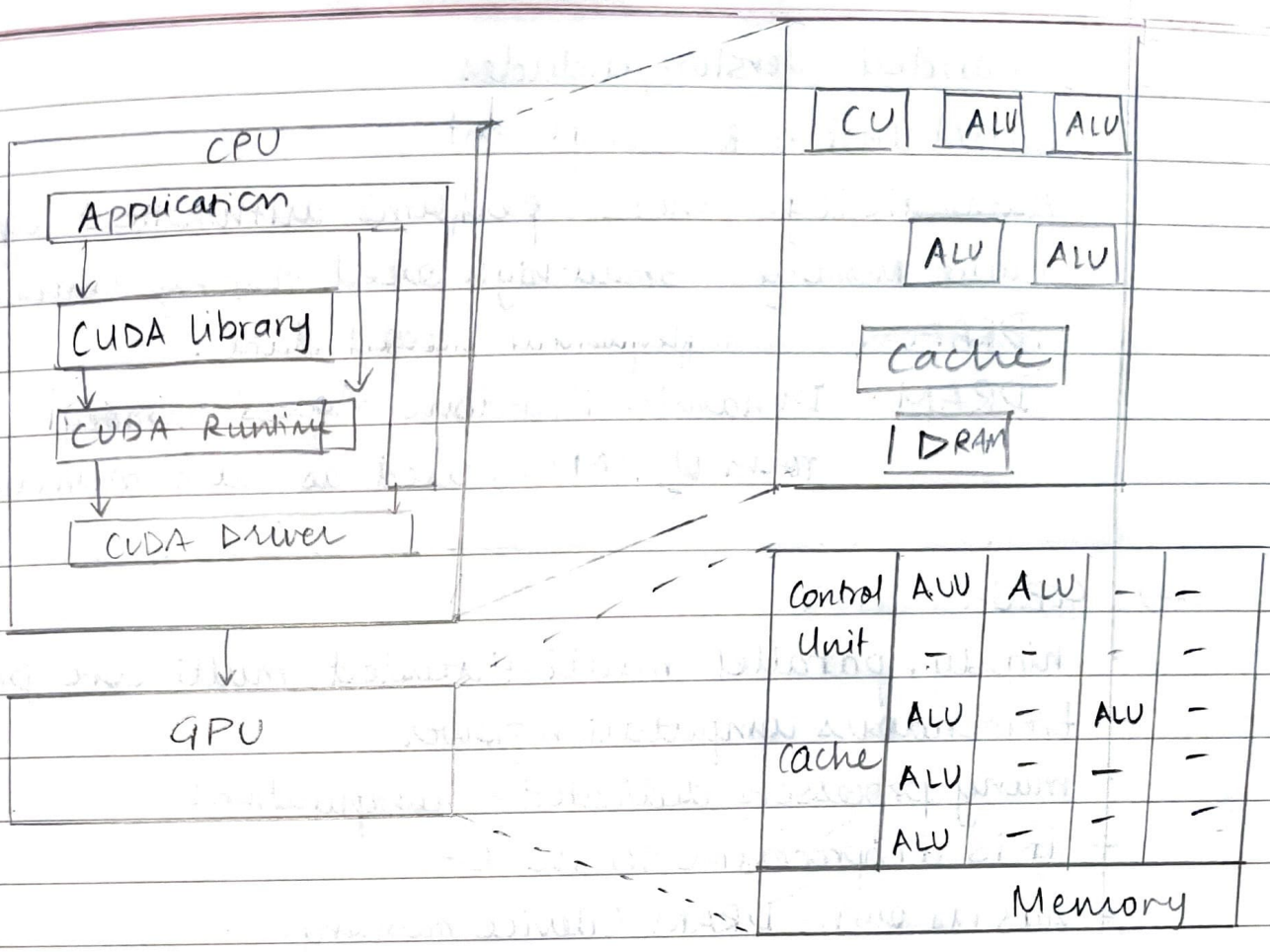
CUDA ARCHITECTURE.

Introduction to CUDA.

- CUDA stands for Compute Unified Device Architecture.
- parallel computing architecture from NVIDIA.
- Using CUDA, GPUs can be used for general purpose processing.
- gives program developers direct access to virtual instruction set & memory of computational elements in CUDA GPUs.
- enables developers to write scalable parallel programs, using a straightforward extension of C.
- works with C, C++ & Fortran.
- accessible to software developers through CUDA accelerated libraries
- is compatible with most standard operating systems.
- provides a way in which we can efficiently process thousands of elements for a task in parallel.
- way that nearby tasks can communicate effectively.
- good for lots of computations & data.

Architecture

- developed by NVIDIA.
- split into two parts: Host & Device
- Host is computer - machine, CPU.
  Device = GPU - hardware.
- for fast single thread execution - CPU is optimized.
  designed to execute one thread or two threads concurrently.
- for high multi-thread exec - GPU is optimized.
  designed to execute many threads concurrently.
- Two main parts - CPU & GPU.



1) CPU.
- host comprises of CPU & memory.
- host code → CPU.
- control intensive tasks are handled.
- small data size tasks are executed.

i) CUDA library.
Advanced libraries that include BLAS, FFT & other functions optimized for CUDA architecture

ii) CUDA Runtime
- Runtime API → higher-level API implemented on top of driver API
- each function of is broken down into more basic operations which are issued to driver API.
- handles dynamic behaviour.

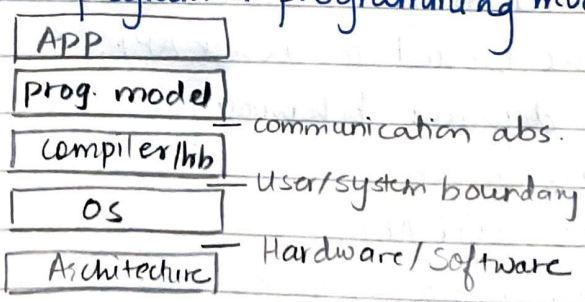
iii) CUDA Driver.
- lower level API.
- relatively hard to program.
- provides more control over how GPU is used.

- Expanded Version includes
  Control Unit : key component
  Arithmetic logic Unit : perfoms arithmetic & logic operations
  Cache Memory : small high speed memory component holding
  ~~DRAM~~ : frequently accesed data.
  DRAM : Dynamic Random Access Memory.
  - form of RAM., used as main memory.

2) GPU.
- highly, parallel, multi-threaded, multi-core processor
- tremendous computation power.
- many processors dedicated to computations.
- it is a coprocessor to the CPU
- has its own DRAM (device memory).
- Device comprises of GPU & its memory.
- Device code is executed by GPU.
- As gpu have high comp. power → comp. intensive tasks handling
- huge amount of data → GPU ✓ CPU ×.
- Expanded versions consists of multiple processors, which give high processing power.

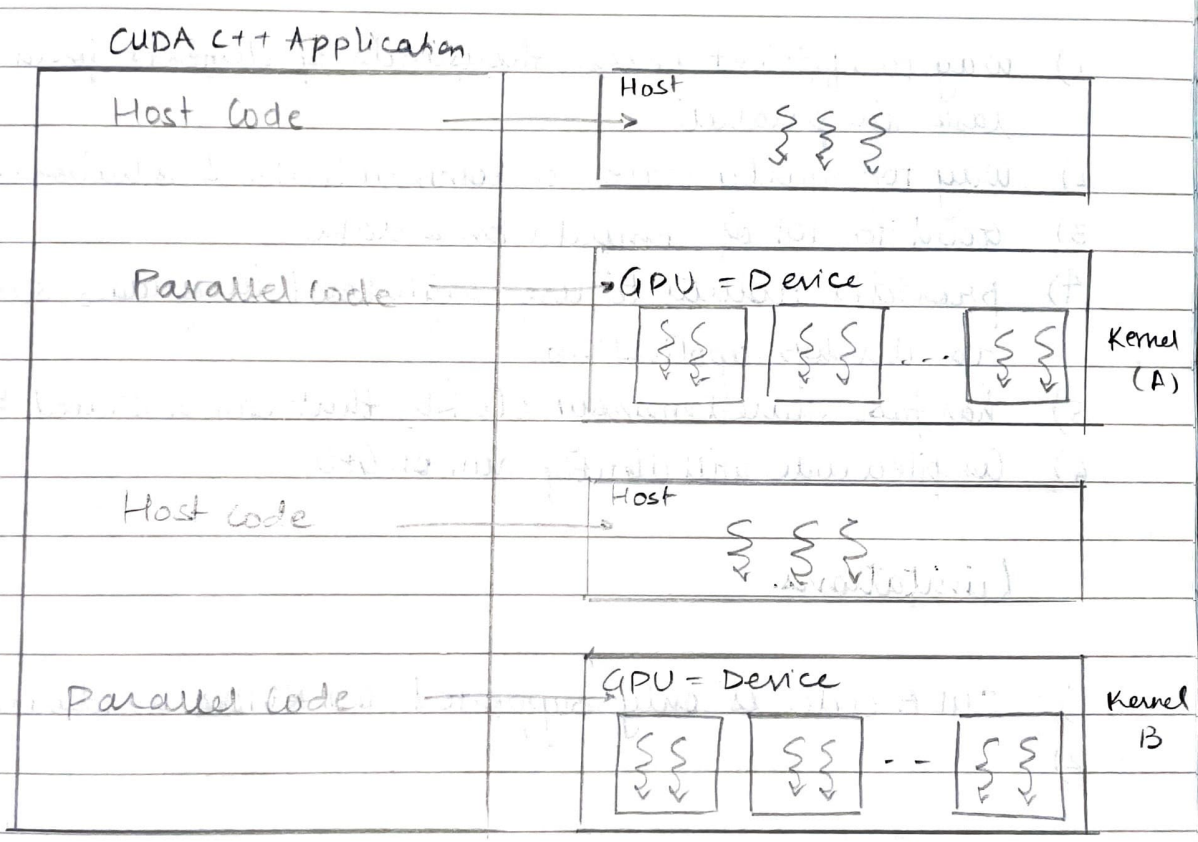
CUDA Programming Model.

- model used for programming acts as a bridge between application & its implementation.
- abstraction bet^n program & programming model ~~arch~~ implem.



CUDA Programming Structure.

- model enables us to execute applications on heterogeneous computing systems.
- combination of serial & parallel execution.
- used to make interaction betⁿ CPU & GPU prog. models.
- data copied from host memory → device memory, results are copied back to host from device



- divides code into Host (CPU) code & Device (GPU) code.
- Serial code is executed in a host thread.
- Parallel code executes in many concurrent devices threads across multiple parallel processors.
- Application executing on hetrogeneous platform is initialized by CPU.
- CPU code manages the enviroment, code & data before loading them to on device.

- Processing flow of CUDA program:

1) Copy data from CPU memory to GPU memory.
2) Invoke kernel to run on GPU.
3) Copy data back from GPU to CPU memory.
4) Release GPU memory & reset GPU.

Advantages.

1) way to efficient process thousands of elements for a particular task in parallel
2) way for nearby tasks to communicate & collaborate efficiently.
3) Good for lot of computation & data
4) provides ability to use high level languages such as C to develop applications
5) has fast shared memory (16 Kb) that can be shared bet" threads
6) Compiled code will directly run on GPU.

Limitations.

1) CUDA code is only supported on NVIDIA hardware.
2)