CUDA Programming

List of Topics

1. Kernel
2. Threads
3. Process
4. Single Instruction Multiple Data Stream (SIMD)
5. GPU (graphics processing units) Memory Hierarchy
   - SRAM, DRAM
   - Shared memory, Constant memory
6. Scheduler
7. Warp
8. Thread Block

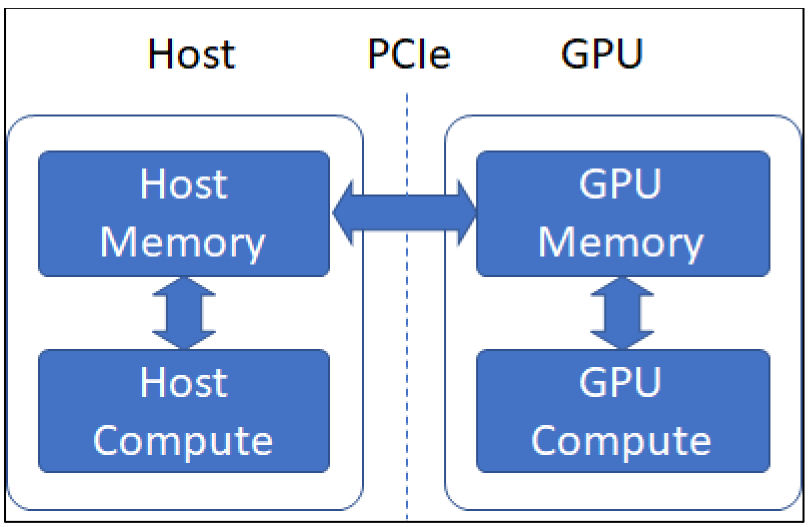
Mohan Sai Naguru

21CS01021

1. Kernel

Host and Device

- The CPU (central processing units) and system's memory are referred to as the host
- The GPU and its memory are referred to as device



What is Kernel

- A function that executes on the device is typically called a Kernel
- In CUDA C, we add __global__ qualifier to standard C

Kernel Call

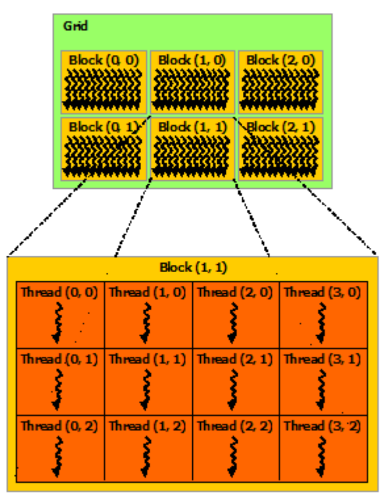
- A Kernel Call is like a regular call in C, except that we embellish it with angular brackets
- <<<blocks,threadsPerBlock >>>
- Passing arguments is similar to standard C
- Sample kernel call

```
#include <iostream>

__global__ void kernel( void ) {
}

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

2. Threads

What is a Thread?

- The lowest level of abstraction for doing a computation or a memory operation in GPU computing
- Organized into thread blocks (or simply called blocks)
- There is a limit on number of threads per block, which is usually 1024
- Threads within a block can communicate with each other through shared memory
- When we use shared memory, it is necessary to synchronize all the threads



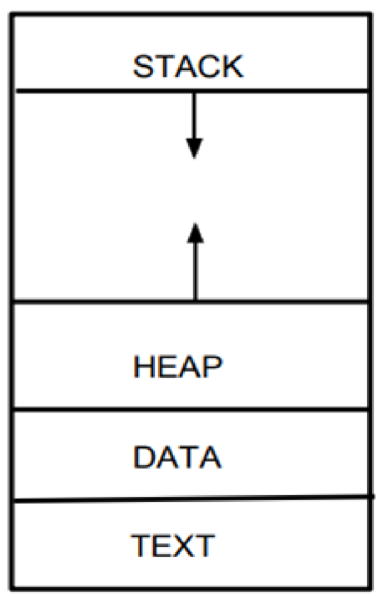
3. Process

- A program in execution

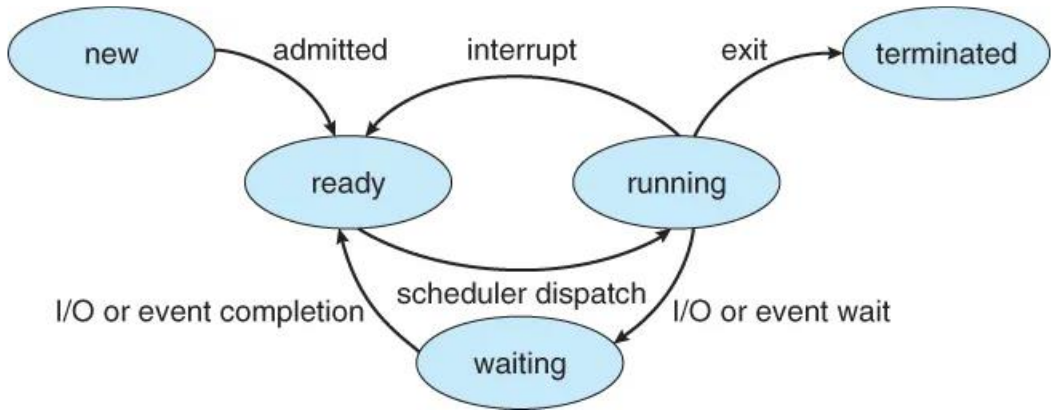
Multiple Parts of Process

- Text Section – The program code
- Program Counter – Current activity
- Stack – Temporary data
- Data Section – Global variables
- Heap – memory dynamically allocated during run time

Memory Layout of Process

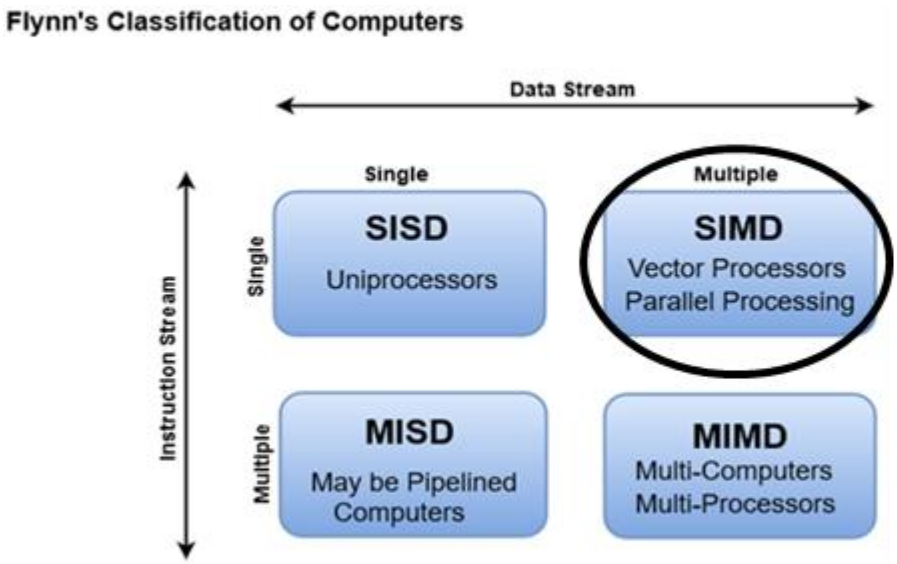


Life Cycle of a Process

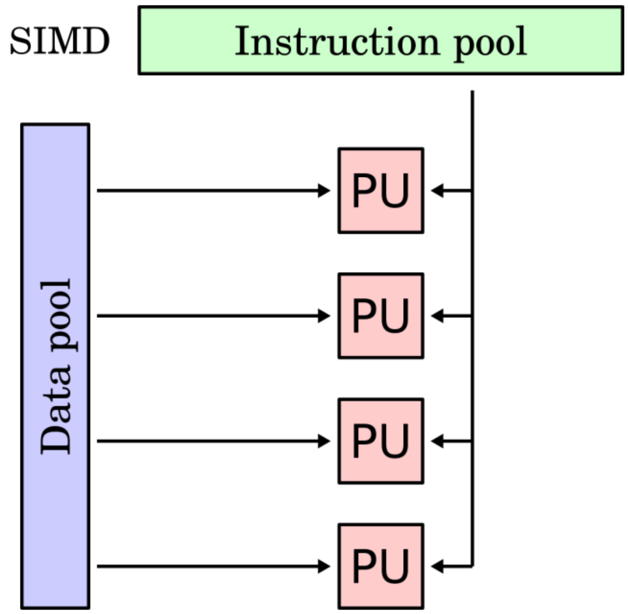


- New state – new process created
- Ready state – ready for computing power allocation
- Running state – process getting executed
- Waiting state or Blocked state – process waiting for the signal (from the user)
- Termination state – process execution completed

4. Single Instruction Multiple Data Stream (SIMD)



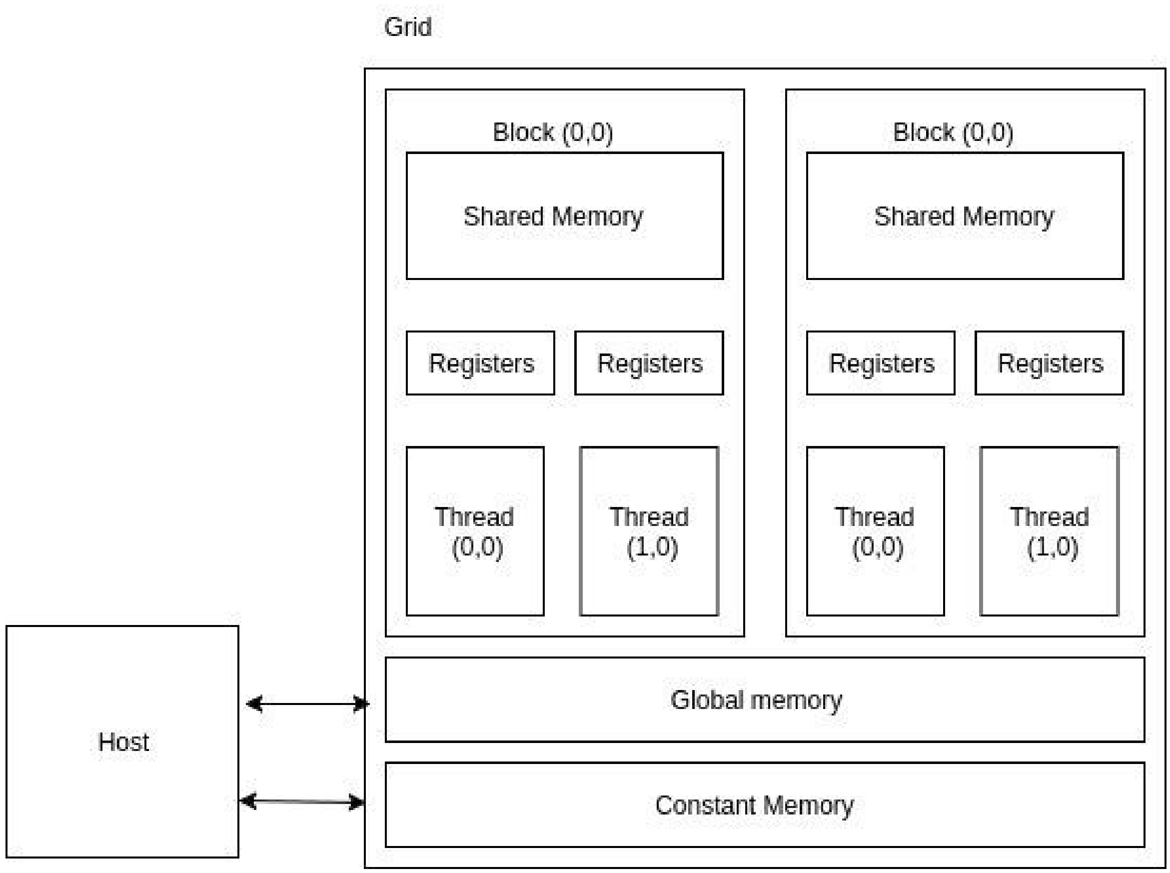
- An organization that includes that includes many processing units under supervision of a common control unit



- There are many commercial implementations of SIMD computing by many companies like Intel SSE, ARM NEON etc.,
- These are basically,
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units (GPU)

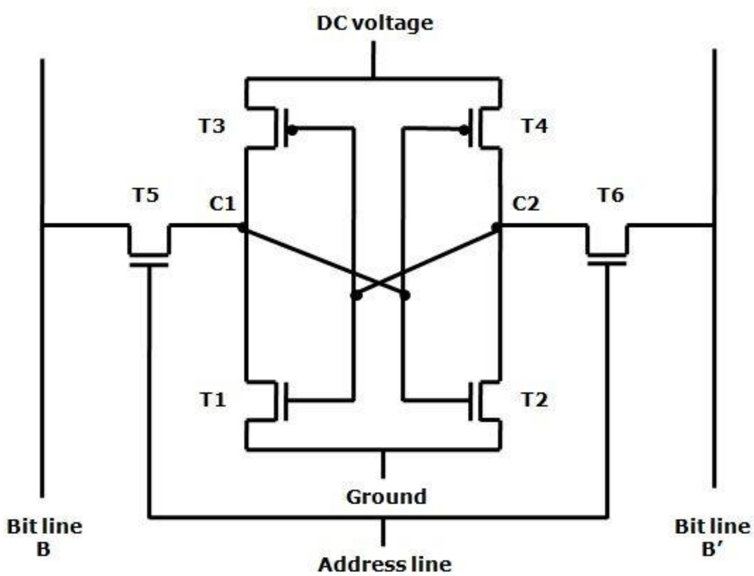
5. GPU Memory Hierarchy

- SRAM (Static RAM)
- DRAM (Dynamic RAM)
- Shared memory
- Constant memory



Static RAM (SRAM)

- Uses latching circuitry (flip-flop) to store each bit
- Volatile memory – data is lost when power is removed
- Usually has 4 or 6 transistors

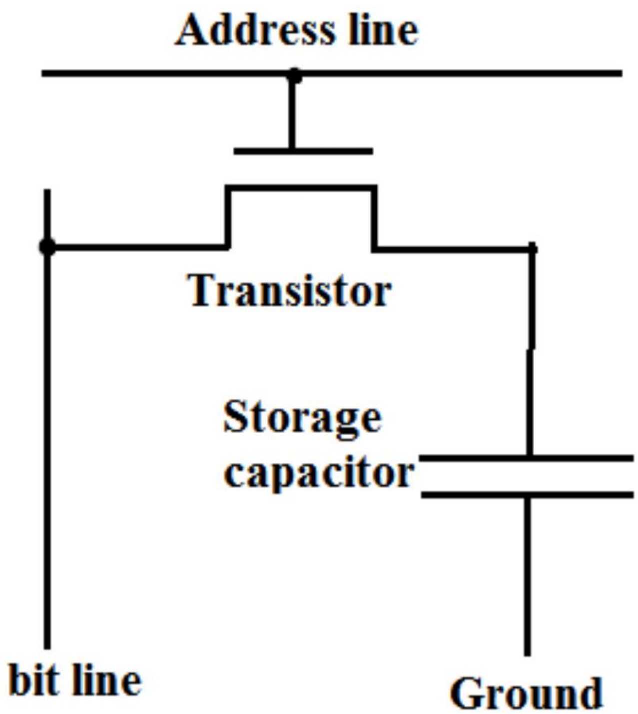


Static RAM (SRAM) Cell

- Used as processors cache memory
- Faster access time

Dynamic RAM (DRAM)

- Uses a capacitor and a transistor to store a bit
- Data is lost as the capacitor is discharged. Hence, it needs to be refreshed every few milliseconds to compensate.
- This in turn, increases the access time



- Referred to as global memory on device, and main system memory on host

SRAM vs DRAM

	SRAM	DRAM
RAM application	Usually used as L2 and L3 cache units in CPU	Used as main memory
Memory capacity	1 MB – 16 MB	4 GB – 16 GB
Cost	Expensive	Cost effective
Placement	On the processor or fixed b/w processor and main memory	On motherboard
Speed	Faster	Slower
Density	Lower	Higher
Power consumption	Lower	Higher
Simplicity	Easy to build interfaces to access memory	More complicated than SRAM
Construction and Design	Complicated (More transistors)	Easier compared to SRAM

Shared Memory

- Variables in shared memory are treated differently than typical variable
- Every thread in a block shares the memory, but threads cannot see or modify the copy of this variable that is seen within other blocks
- Shared memory buffers reside physically on GPU as opposed to residing in the off-chip DRAM
- In turn, it decreases the latency of shared memory buffers

Synchronization

- Let us assume that thread A writes a value to shared memory, and we want thread B to do something with the value. In that case, we cannot have thread B start its work until we know the write from thread A is complete.
- To make sure that synchronization is needed

Constant Memory

- Small amount of memory that is stored on the GPU's on-chip memory
- Read-only memory and shared across all threads in a CUDA kernel
- Like global memory in that, all threads can access it, but it has some significant differences

Advantages

- Faster than global memory, as it is stored in a cache on GPU – lower latency, reduces the required memory bandwidth

(Memory bandwidth – rate at which data can be read from or stored into)

- Optimized for read-only access

Disadvantages

- A half-warp is allowed to place only a single read request at a time (refer to Warp later in the file)

6. Scheduler (OS)

Process Scheduling

- An important part of multi-programming OS
- It is the process of removing a running task and selecting another task for processing into different states like ready, waiting, and running

Process Scheduling Queues

- The OS maintains a separate queue for each state along with the process control blocks (PCB) of all processes.
- The PCB moves to a new state queue, after being unlinked from its current queue, when the state of a process changes.
- Job queue – makes sure processes stay in the system
- Ready queue – set of all processes in main memory ready and waiting for execution
- Device queue – processes blocked due to unavailability of I/O devices

Process Schedulers

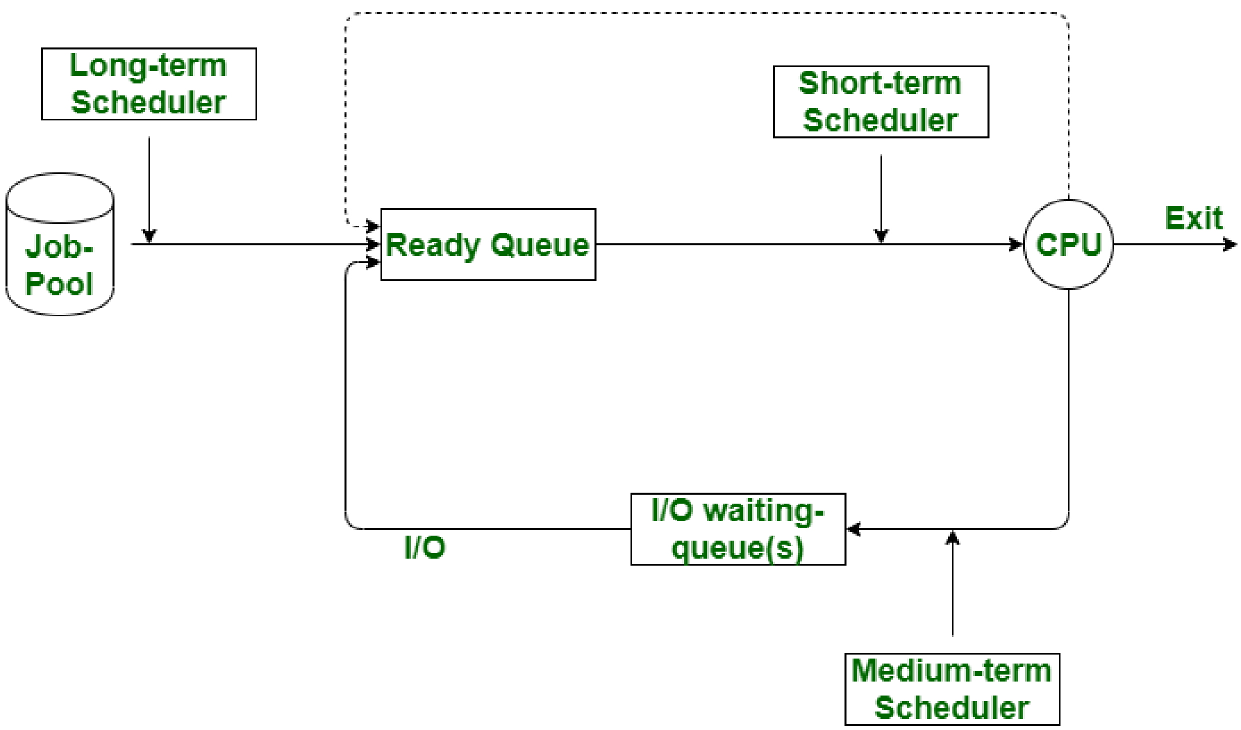
- Special type of system software (part of a kernel) that handle process scheduling
- Selects the jobs that are to be submitted into the system and decides whether the currently running process should be kept running or not
- If it decided that currently running process should not keep running, then it decides which process should run next

Types of Process Schedulers

- Long Term Scheduler
- Short Term Scheduler
- Medium Term Scheduler

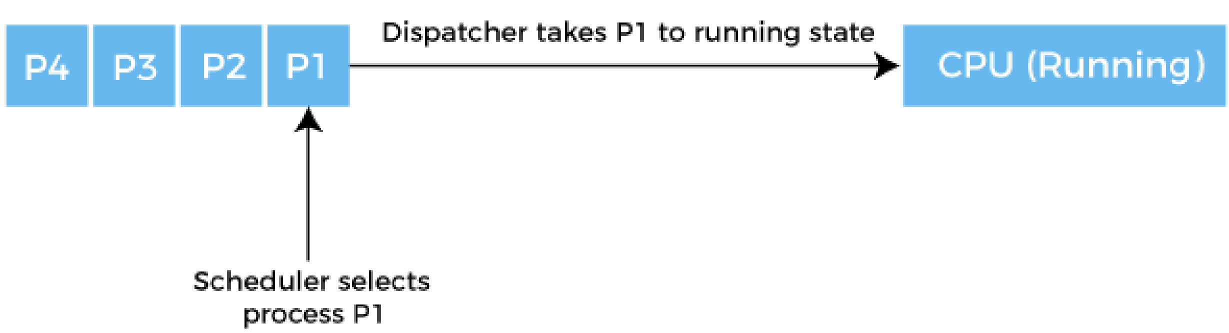
Long-Term Scheduler

- Also known as job scheduler
- Determines which process should be entered into system for processing
- It selects and loads the processes into the memory for execution with the help of CPU scheduling
- Many systems like time-sharing OS, do not have a long-term scheduler as it is only required when a process changes its state from new to ready.



Short-Term Scheduler

- Also known as CPU scheduler
- Increases system performance as per chosen set of criteria
- Change of ready state into running state
- Selects a process from multiple processes in the ready state to execute it and allocates CPU to one of them
- Faster than Long-term schedulers
- Also called a dispatcher as it makes the decision on which process will be executed next



Medium-Term Scheduler

- Removes processes from the memory and is a part of swapping
- Reduces the degree of multi-programming (by swapping out a subset of processes)
- In charge of handling the swapped-out processes
- When a running process makes an I/O request, it is suspended (it cannot be completed). Thus, to remove the process from memory and make space for others, the suspended process is sent to secondary storage by the medium-term scheduler

Comparision

S.No.	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
1.	A job scheduler	A CPU scheduler	A process swapping scheduler
2.	Slowest speed	Fastest Speed	Speed is between the other two
3.	Controls the degree of multiprogramming	Provides less control over the degree of multiprogramming	Reduces the degree of multiprogramming
4.	Absent or minimal in the time-sharing OS	Minimal in time-sharing OS	Part of time-sharing OS
5.	Selects a process from pool and loads it into memory for execution	Selects a process that is ready for execution	Re-introduces processes into memory for continued execution

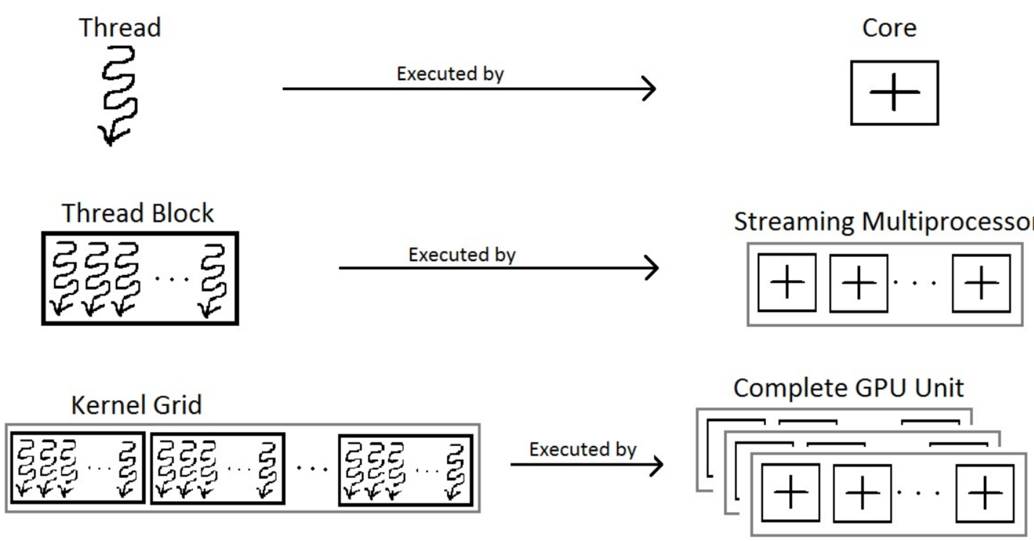
7. Warp

- A collection of 32 threads that are woven together
- At every line in the program, each thread in a warp executes the same instruction on different data
- When it comes to handling constant memory, NVIDIA hardware can broadcast single memory read to each half-warp
- If every thread in a half-warp requests data from the same address in constant memory, GPU will generate only a single read request and broadcast the data to every thread
- If a lot of memory is being read from same address in constant memory, only $1/16^{th}$ of memory traffic is generated compared to when using global memory
- Although it can dramatically increase performance when reading from the same address, it slows performance when reading

from different addresses from constant memory

8. Thread Block

- A programming abstraction that represents a group of threads that can be execute serially or in parallel
- For better process and data mapping, threads are grouped into thread blocks
- Also simply called block
- The maximum number of threads per block usually is 1024 (depending on the GPU architecture)
- Multiple blocks are combined to form a grid. All the blocks in the same grid contain same number of threads per block



- The threads in the same block run on the same streaming multiprocessor

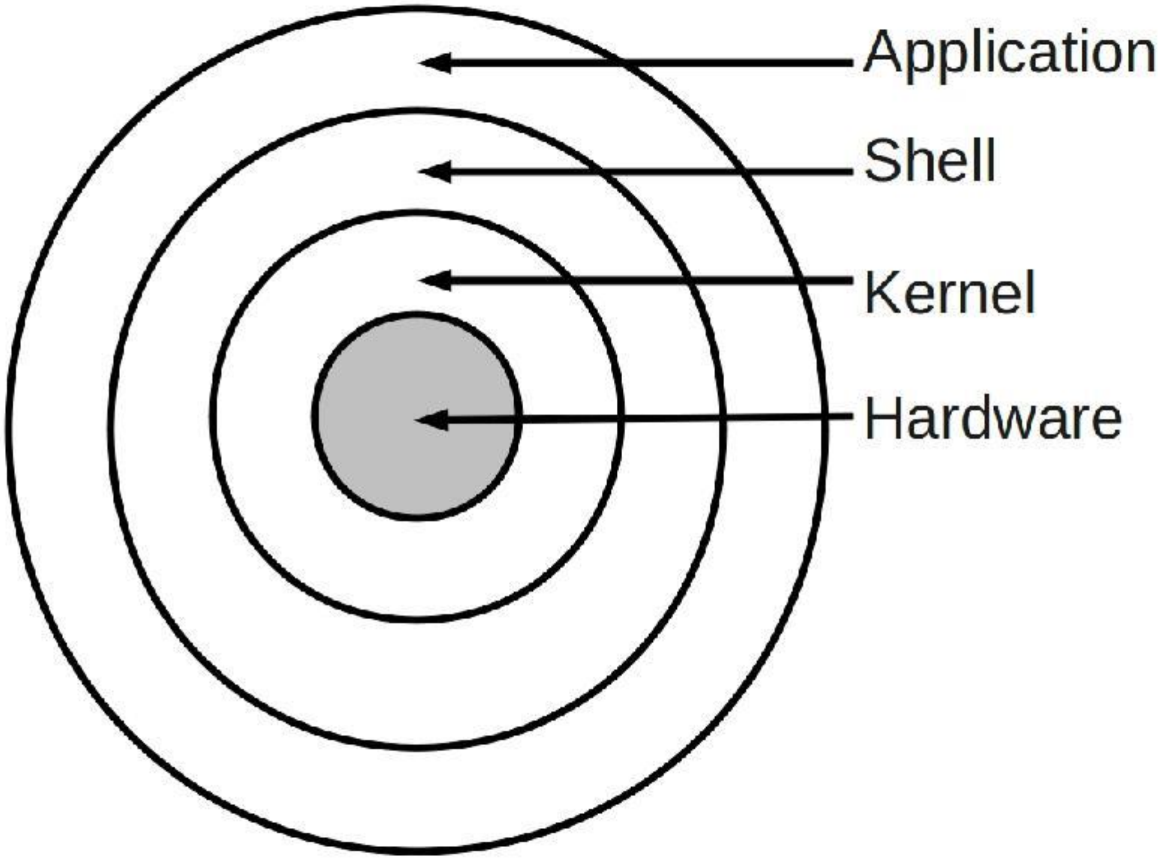
Streaming Multiprocessor (SM)

- Each architecture in GPU consists of several SM or Streaming Multiprocessors
- SMs are general-purpose processors with a low clock rate target and a small cache
- The primary task of an SM is that it must execute several thread blocks in parallel
- As soon as one of its thread blocks has completed execution, it takes up the serially next thread block
- To achieve this, and SM contains the following
  - Execution cores (SP)
  - Caches
  - Schedulers for warp
  - A substantial number of registers
- Usually, one SM can handle multiple thread blocks at the same time

Additional

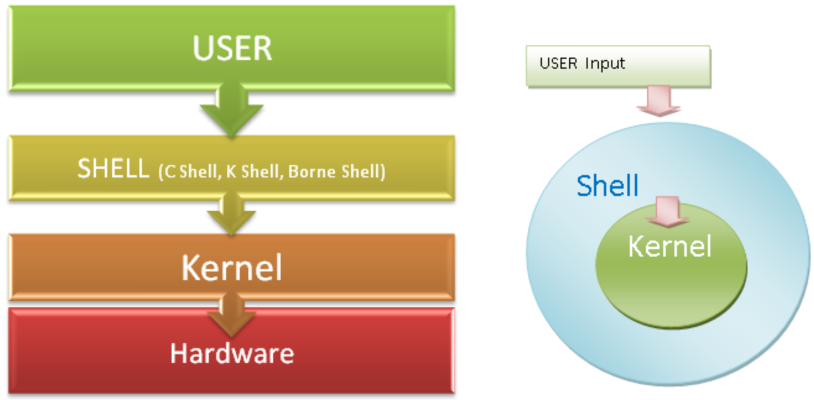
- A few terms like ‘kernel’ and ‘threads’ have similar meanings in the context of the Operating System and the context of CUDA programming but are different.
- The terms ‘kernel’ and ‘threads’ have been explained in the context of CUDA programming in the main text.
- Their meanings in the context of OS have been explained below.
- Similarly, Schedulers are different for CPU processes and on GPU, Schedulers for CPU processes have been explained in the main text but in case of GPU, we have warp schedulers.

1. Kernel



What is Kernel

- core component of an OS
- acts as a bridge between applications and data processing at the hardware level using inter-process communication and system calls.



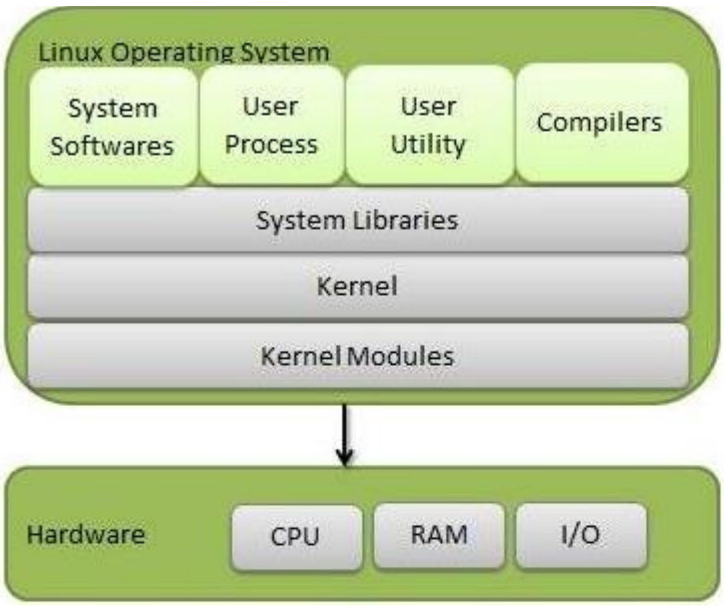
What does it do?

- Establish a connection between user level application and hardware.
- Decide the state of an incoming process
- Disk management
- Memory management
- Task management

Kernel vs OS

- OS is a complete software package that includes a kernel and other system-level components such as device drivers, system libraries, and utilities.
- OS also provides a higher-level interface to the user, such as the GUI, command-line interface,

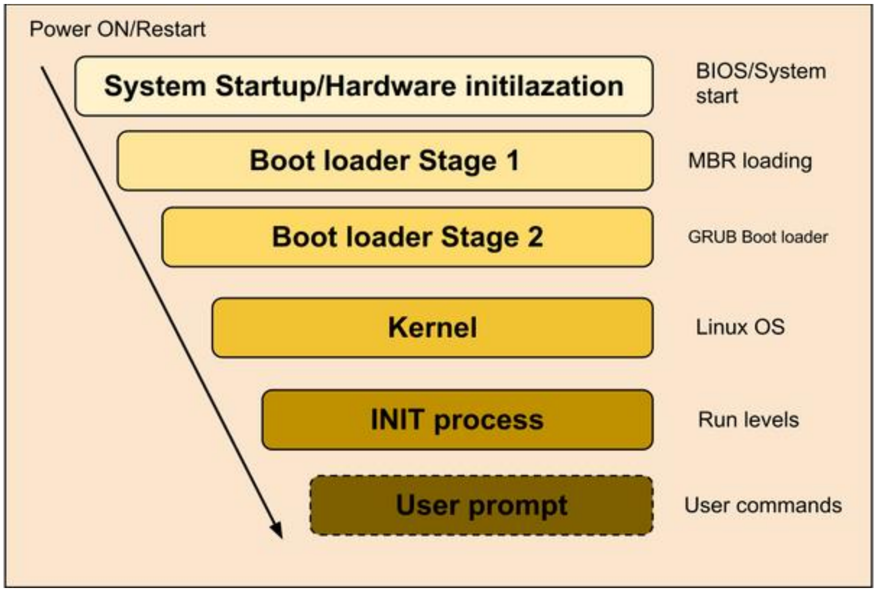
and file system. In contrast, the kernel provides low-level services like memory, process, and device management to other parts of the operating system.



Kernel and Bootstrapping

The following are the stages involved in Bootstrapping

- Firmware (BIOS) initialization
  - Initialize the essential hardware
- Bootloader
  - Loads OS kernel into memory
- Kernel
  - initializes hardware components, sets up the system's memory map, and prepares for the transition from kernel mode to user mode.
  - Launches init process (pid = 1)
- Init process
  - The init process sets up the user environment, initializes system services, and starts user-level processes.

- 

- User prompt
  - Displays information on the screen and asks the user to provide information, choose an option, or make a decision.

2. Threads

```cpp
#include <iostream>
#include <thread>

using namespace std;

static const int nt=10;

void Hello(int num)
{
    cout<<num<<": Hello thread!"<<endl;
}

int main()
{
    thread t[nt];

    for (int i = 0; i < nt; ++i)
    {
        t[i]=thread(Hello,i);
    }

    for (int i = 0; i < nt; ++i)
    {
        t[i].join();
    }

    return 0;
}
```

```
:~/Playground
:~/Playground$ g++ -std=c++11 -pthread thread.cpp -o thread && ./thread
2: Hello thread!1
4: Hello thread!
: Hello thread!
5: Hello thread!
0: Hello thread!
6: Hello thread!
7: Hello thread!
3: Hello thread!
9: Hello thread!
8: Hello thread!
:~/Playground$
```

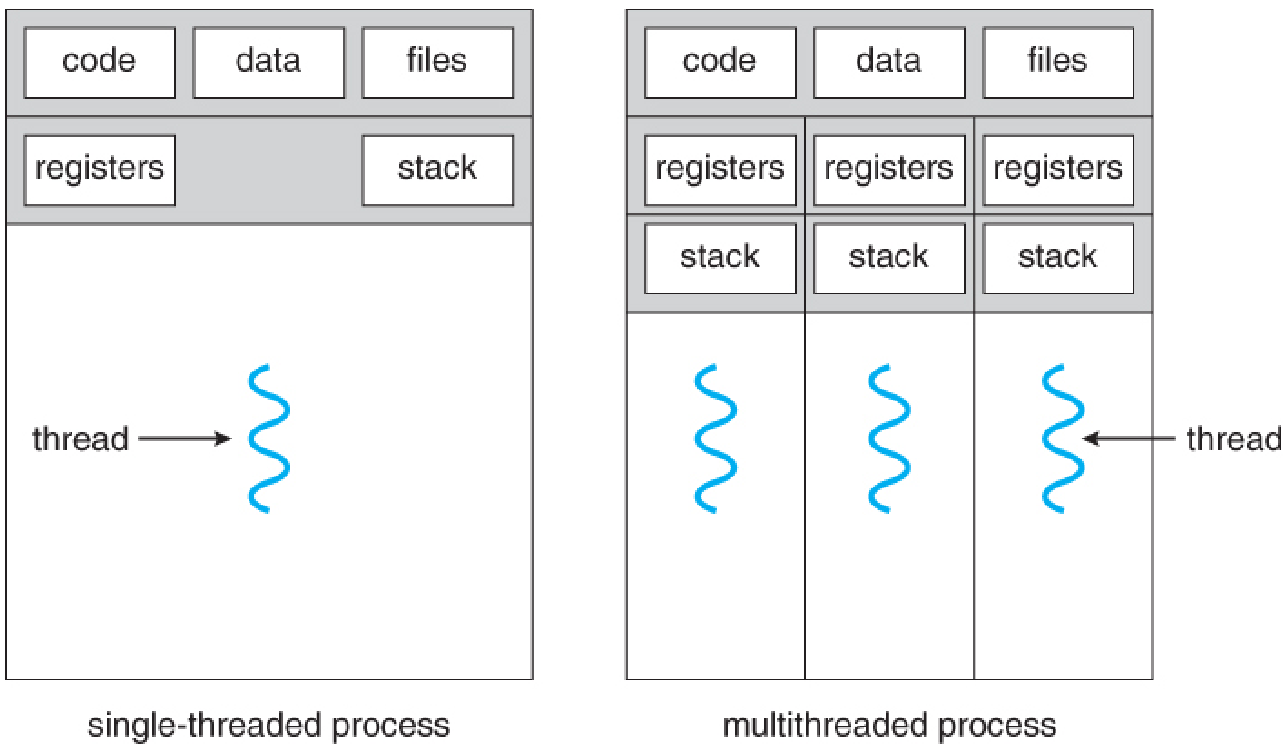
Multithreaded Parallel Execution Example in C++

What are Threads

- separate execution path within a program
- light-weight processes that OS can schedule and run concurrently with other threads

- share memory and resources as the program that created them

Single-threaded vs Multiple-threaded process



Why Multi-threading?

- to achieve parallelism by dividing the process into multiple threads

Advantages

- Responsiveness
- Faster context switch
- Effective utilization of multi-processor system
- Resource sharing
- Easier communication
- Enhanced through-put of the system

Types of Threads

There are two types of threads

- User Level Thread
- Kernel Level Thread

User-Level Threads

- Not created using system calls
- Kernel has no work in management of user-level threads
- Can be easily implemented by the user

Advantages

- Easier implementation
- Faster context switch
- More efficient
- Simple representation (Program Counter, Register Set, and Stack Space)

Disadvantages

- Lack of coordination between Thread and Kernel
- In case of a page fault, the complete process can be blocked

Kernel-Level Threads

- System calls are used to generate and manage these threads
- The OS Kernel helps in the management of kernel-level threads
- Complex implementation

Advantages

- Has up-to-date information on all threads (has its own thread table)
- Applications that block frequency (meaning applications that perform highly CPU-intensive) are to be handled by Kernel-level threads
- Whenever any process requires more time to process, Kernel-level threads provide more time for it.

Disadvantages

- Slower than user-level threads
- Complex implementation