

GPU Architectures and Programming
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture # 42
OpenCL - Runtime System (Contd.)

(Refer Slide Time: 00:33)

Page 1/1

OpenCL Execution Model

(Figure from reference[1])

OpenCL - Runtime System Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Hi, welcome back to the lectures on GPU architectures and programming. So, in the last lecture, we have briefly introduced these concepts of OpenCL programs and we were discussing the OpenCL execution model.

(Refer Slide Time: 00:40)

Page 1/1

OpenCL Execution Model: Context

A context is an abstract container that exists on the host
The context includes the following resources:

- ▶ **Devices:** The collection of OpenCL devices to be used by the host
- ▶ **Kernels:** The OpenCL functions that run on OpenCL devices
- ▶ **Program Objects:** The program source and executables that implement the kernels
- ▶ **Memory Objects:** A set of memory objects visible to the host and the OpenCL devices.

OpenCL - Runtime System Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

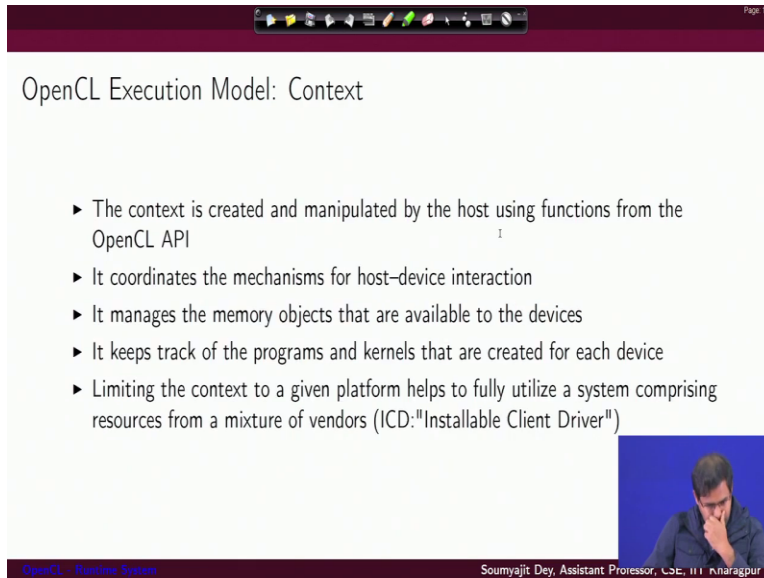
So, continuing that will slowly introduce the different OpenCL concepts one by one. So, what will come first is the OpenCL context. So, we are starting with the execution model of programs. So, what is the context? So, essentially our context is nothing but an abstract container that exists in the host side and it includes the following resource definitions. So it is like object we said an abstract object and abstract definition and its context containing information the following resources.

That is first of all the devices that it will contain the information about what are the OpenCL devices which had to be used by the host and how to access them, it will contain the different OpenCL functions that are to be run on the respective OpenCL devices and it will contain this program objects. So essentially what our program is the program source and executables that implement the kernel, so you write a kernel.

Now, from that kernel, the OpenCL runtime system will build a program object and then it will also contain the memory objects. So the set of memory objects which are visible to the host and OpenCL devices. So in a nutshell, so we are trying to define how OpenCL programs execution is set up. It is all based on this definition of a context and the context captures what am I going to execute, which are the kernels, which are translated into a program objects.

And they are going to use data from again the memory model, which is which is kind of known to the host in the form of memory objects and the devices like which are the actual assembly architectures where these kernels will be executing.

(Refer Slide Time: 02:34)



OpenCL Execution Model: Context

- ▶ The context is created and manipulated by the host using functions from the OpenCL API
- ▶ It coordinates the mechanisms for host-device interaction
- ▶ It manages the memory objects that are available to the devices
- ▶ It keeps track of the programs and kernels that are created for each device
- ▶ Limiting the context to a given platform helps to fully utilize a system comprising resources from a mixture of vendors (ICD: "Installable Client Driver")

OpenCL - Runtime System Soumyajit Dey, Assistant Professor, IIT Kharagpur

So, a context is something that gets created by the OpenCL API I mean, as essentially, you have to write a suitable host program, and you have to call suitable functions from the underlying OpenCL API through which have to come we have to actually create the context. Now, it is useful primarily for executing the kernels getting their results back from the devices using those devices for executing more kernels in other devices and all that.

So, essentially the context inside I mean, whatever coordination one will do, I mean, we understand from our basic knowledge and GPU programming now, that the host device is actually responsible for orchestrating the execution of kernels in different GPU devices. So, coming to OpenCL, the similar concepts are actually going to hold. So, the host problem is the host site code is actually orchestrating the execution of this kernel objects on different devices.

But all these things are happening inside the definition of this context is also managing the memory objects it is deciding which memory object is accessible to which device and all that so, essentially the context is an abstraction which binds devices with programs and memory objects. Now, limiting the context to a given platform helps to fully utilize a system comprising resources from a mixture of vendors this is a useful thing.

Like why do we need the definition of a context, the primary reason is because of the OpenCL platform model. So, it may always have been that you have built a large compute system for

multiple different kinds of devices coming from different vendors, let us say MD, let us say somebody else, they are all connected to the PCI Express Bus. Now let us understand that each of them may be OpenCL vendors.

So what they will do is each of them will provide the underlying ICD Installable Client Driver conforming to the OpenCL specification, but they are all specific for each vendor. So a collection of devices from a specific vendor for them. If I define a context, then that context will use that vendor applied ICD for performing your execution. Some other context I will define for some other diverse set of devices from some other vendor and I can have the host program, coordinate executions in different devices, where each of the devices are mapped to different context.


So, inside the context, I have the memory definition that device definition pro kernel definition and program execution going on I can have another context where another set of kernels are executing and top level program you can actually manage multiple contexts executing on different vendor server supplied platforms. And it can help these different contexts communicate data among each other for performance benefits are getting more terrorization.

So the basic idea would be that you want to have an obstruction through which you can actually link up things from different vendors in a coherent way, and still proceed with billing something big.

(Refer Slide Time: 06:06)

OpenCL Execution Model: Command Queue

- ▶ Host creates a data structure called command-queue to co-ordinate execution of the kernels on the devices.
- ▶ Host places commands into the command-queue which are then scheduled onto the devices within the context.
 - ▶ Kernel execution commands
 - ▶ Memory commands
 - ▶ Synchronization commands
- ▶ These execute asynchronously between the host and the device
 - ▶ In-order Execution (default)
 - ▶ Out-of-order Execution



OpenCL - Basics Edition Soumyajit Dey, Assistant Professor, IISc, Bangalore

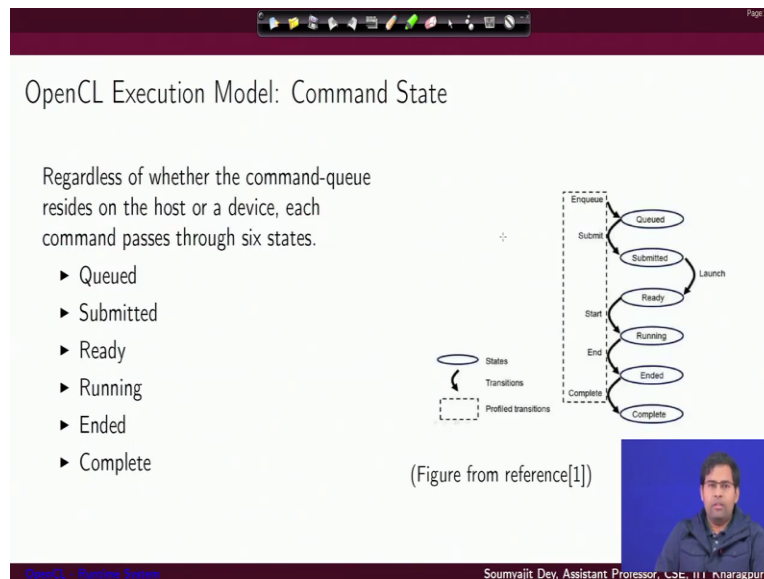
The next important concept here will be what we call as the command queue. So when we are talking about context, as I said, it is basically an abstraction through which I am trying to define how an OpenCL program, we look at the memory, how opens CL program, you look at a device and all that but there has to be a specific formalization that the devices are going to execute this kind of commands in this specific order.

And how will that sequence of commands coordinate among each other? So for that, the top level host let us understand that difficulty with the host, the host is trying to manage a set of devices. As I said in the worst case that devices can be of different vendors. That problem is solved by bunching out devices together into a set of context. So the host has to manage a set of devices sitting in different contexts. Well, that is fine. But when the host is trying to dispatch some action for each of the devices, the host needs some mechanism for doing that.

So what the host does is it creates our data and other data structure called command queue through which it can submit instructions that to our device that you execute a kernel, you get this data back, you use this data as an input. So these are all elements the actions which are kernel will perform, which a device will perform, and in what order it will perform that is orchestrated by the host through a command queue. So all the hosts does is that it replace commands to the command queue, which are then scheduled into the devices within the context.

A device will have its own command queue and the kinds of commands can be different. There can be kernel execution commands, they can be memory transfer related commands for the device and they can also be synchronization commands. Now, these queues can execute both in order as well as out of order. The host can also specify that for some device, whether the commands that are queued up in the command queue of the device whether they will execute in the order in which they are queued up or they can execute out of order.

(Refer Slide Time: 08:22)



Now, so, we have understood what is the command so, essentially it is one of these following things like kernel execution and memory transfer instruction or a synchronization command. So, these are the different possible commands that our hosts can issue to a specific kind of device. Now, once this command is issued, it can go through the following well defined states that the OpenCL runtime system understands.

So, a command will be queued. The command will be submitted from the device it will be de-queued and submitted to the device. It is ready to execute. It is actually running the command ends and the command gets completed. So these are all the different possible status messages for a command under execute, which is the inner from the lifetime when it start sitting in a queue up to the point when it gets completed in its execution.

(Refer Slide Time: 09:18)

The screenshot shows a presentation slide with a dark red header and footer. The title is 'OpenCL Execution Model: Synchronization'. The main text defines synchronization as mechanisms that constrain the order of execution between two or more units of execution. It lists two types: 'Work-group synchronization' (constraints on work-items in a single work-group) and 'Command synchronization' (constraints on the order of commands launched for execution). A small video inset in the bottom right shows a man speaking. The footer contains the text 'OpenCL - Runtime Execution' and 'Soumyajit Dey, Assistant Professor, IIT Madras'.

OpenCL Execution Model: Synchronization

Synchronization refers to mechanisms that constrain the order of execution between two or more units of execution.

- ▶ **Work-group synchronization:** Constraints on the order of execution for work-items in a single work-group
- ▶ **Command synchronization:** Constraints on the order of commands launched for execution

OpenCL - Runtime Execution Soumyajit Dey, Assistant Professor, IIT Madras

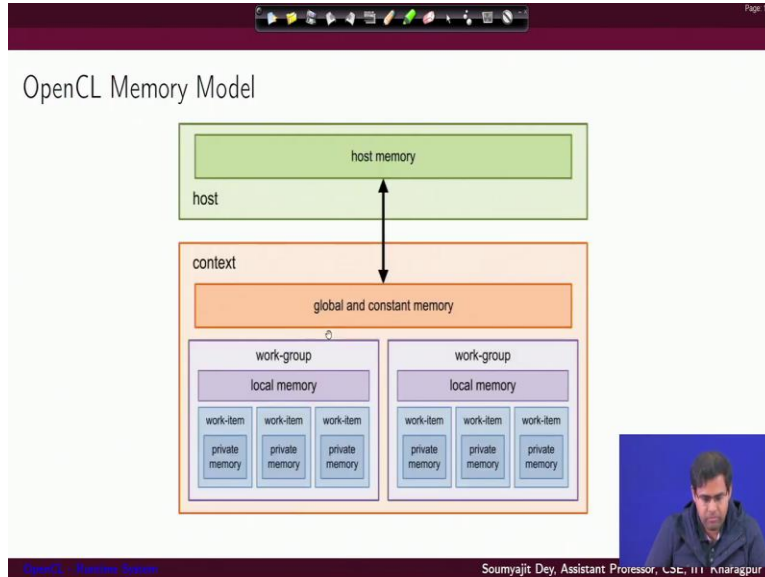
Now in OpenCL synchronization execution model, the other important thing from our perspective would be synchronization. So, what is synchronization? So, we understand that in our data program, we often will require threads to synchronize at certain specific points, which are the barriers right for example, in CUDA we had the synchronized constraints, similar, you know, OpenCL like, we have thread blocks are while groups containing execution of work items and work items will often like to synchronize among each other.

So, the synchronization would refer in this case to mechanisms that constraint the order of execution between two or more units of execution. So it is possible to do while group synchronization. So this essentially means constraints on the order of execution for work items in a single work. So this is kind of similar to idea of sync threads in CUDA programs. So a sync thread would synchronize all the threads inside a thread block.

But not across thread blocks. Here in this case, I can perform in similar way I can synchronize work items sitting inside single work group. The other notion of synchronization is command synchronization. So I can put sensors synchronization primitives inside the command queue. As you can see that this is one of the possible commands that can be submitted by a host program in a command queue. So I can give a kernel execution command I can be transported at command and I can also give a synchronization command.

So when we talk about a synchronization common that would mean a constraints on the order of commands launched first in execution. So, that would mean a suitable synchronization command will be enqueue by the host on the command queue. And it will restrict the order in which commands are going to be launched for execution.

(Refer Slide Time: 11:15)



Now, as we have been discussing that in OpenCL, we define that execution model and also a memory model. And so, this is a picture of the memory model. So, you will have host memory and so, that is a host side memory and host side compute environment, which is we have here from the host, you can copy data to the global memory of a specific context. Now, this is where things become a bit more complex and different with respect to CUDA.

We need not have a single device we need not have a single GPU, I can have a rather small collection of devices, I can among the collection of devices I can have devices could be coming from different vendors accordingly. I What I will do as a memory object, I will define a context as an abstraction inside this current context, I will have a collection of devices they are global and constant memory. And I will have this notion of work groups executing on the processing elements.

So, while group will be constituting work items, and this work items will be have access to their private memory, the shared local memory and in the worst case, they will all have access to the

global memory in the context. So, I can have multiple devices in the context. So I can have a set of 4 groups executing together in one device, another set of 4 groups including in other device like that. So, to be more specific, what we have more here is the host can orchestrate execution across develop multiple devices and they will set up multiple devices can begin group into this abstraction of contexts.

(Refer Slide Time: 13:04)

Memory Objects

The contents of global memory are memory objects. Use the OpenCL type `cl_mem`.

- ▶ **Buffer:** Stored as a block of contiguous memory and used as a general purpose object to hold built in types (such as `int`, `float`), vector types, or user-defined data used in an OpenCL program. Can be manipulated through pointers much as one would with any block of memory in C.
- ▶ **Image:** Holds one, two or three dimensional images. Formats are based on the standard image formats used in graphics applications. Must be managed by functions defined in the OpenCL API.

OpenCL - Runtime System Soumyajit Dey, Assistant Professor, IIT, IIIT Anaragpur

Now, coming to the memory object definitions, so, the contents of global memory are essentially open memory objects with used OpenCL type CL underscore name. Now, memory objects can be of the type buffered, which are stored at block of contiguous memory and used as general purpose object to hold the built in types like integers or floats, other vector types or any user defined data type in an OpenCL program. And they can be manipulated light through pointers similar to any kind of memory block access that can you can do in normal C program.

The other thing is image so in OpenCL, you have a separate memory type defined for images, which is useful for holding multi-dimensional images up to 3 dimension. And the formats are based on standard image formats. So like you have support for different standard image format. So that is specifically provided targeting graphics applications, which was the original workload domain that was targeted. And they need to be managed by functions which are defined in the OpenCL API.

Which would mean whatever vendor provided ICD file you get as part of the OpenCL distribution on your browser. It should contain implementations of the required functions for reading and storing such image objects.

(Refer Slide Time: 14:28)

OpenCL Memory Model

	Global	Constant	Local	Private
Host	Dynamic Allocation	Dynamic Allocation	Dynamic Allocation	No Allocation
	Read/Write access to buffers and images	Read/Write access	No access	No access
Kernel	Static Allocation for program scope variables	Static Allocation	Static Allocation.	Static Allocation.
	Read/Write access	Read-only access	Read/Write access	Read/Write access

OpenCL - Runtime System Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

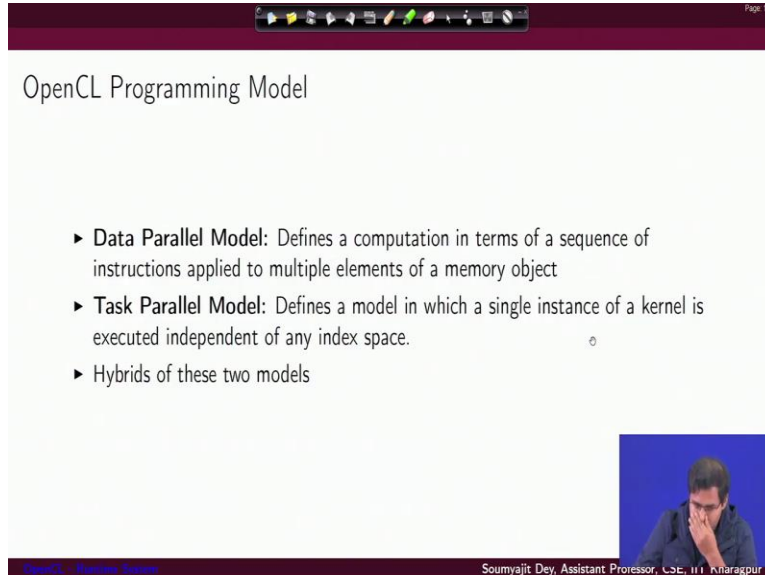
So if we do a summary here of the OpenCL memory model, so from the global perspective, in the host, you have support for dynamic allocation. You have read/write access to buffers and images. From the kernels perspective. You have static allocation for programs called variables and read/write access. If you look at the constant memory segment, again, from the host side, you can do a dynamic allocation. You have read/write access from the kernel side, you would have static allocation and only read only access from the constant.

But, so whatever has to be defined from the host side as constant. Now, from the local memory part again for host you have a dynamic allocation. But from the kernel side you again have static allocation, but every thread can of course, read as well as write in its local the shared local memory of each of the compute units and have similar behavior is also expected from the private memory for each thread.

So, that is what you also have read/write access in the private memories, of course, for the local and the private memory which are in the device, because this is in the inside each of the compute units and this is inside each of the processing elements that are sitting inside the compute units.

So they are all in that device. So, the host really does not have any access to this memory segments. This is what is supported by the OpenCL kernels that are submitted to the respective devices.

(Refer Slide Time: 15:57)



The screenshot shows a presentation slide with a dark red header and footer. The title is "OpenCL Programming Model". The main content is a bulleted list of three programming models. In the bottom right corner, there is a small video inset showing a man with glasses and a dark shirt, looking thoughtful with his hand to his chin. The footer contains the text "OpenCL - Runtime System" and "Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur".

OpenCL Programming Model

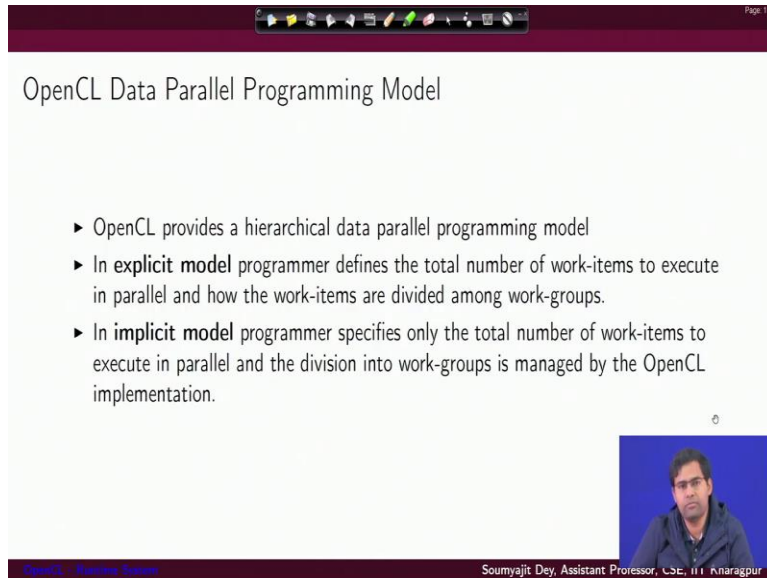
- ▶ **Data Parallel Model:** Defines a computation in terms of a sequence of instructions applied to multiple elements of a memory object
- ▶ **Task Parallel Model:** Defines a model in which a single instance of a kernel is executed independent of any index space.
- ▶ Hybrids of these two models

OpenCL - Runtime System

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, if we come to the programming model, so you have a data parallel model, of kernel execution where you define the computation in terms of a sequence of instructions that are, of course applied to multiple elements of a memory object. So that is how we have been writing usual C code and using usual CUDA code. And the alternative is task parallel model. Where do you define a model in which single instance of a kernel is executed independent of any index space. So this is also something we will look into and of course, you have support for a hybrid combination of both these models.

(Refer Slide Time: 16:36)



The screenshot shows a presentation slide with a dark red header and footer. The title is 'OpenCL Data Parallel Programming Model'. The main content is a bulleted list. At the bottom right, there is a small video inset of a man speaking. The footer contains the text 'OpenCL - Basics Overview' and 'Soumyajit Dey, Assistant Professor, CSE, IIT Annapur'.

OpenCL Data Parallel Programming Model

- ▶ OpenCL provides a hierarchical data parallel programming model
- ▶ In **explicit model** programmer defines the total number of work-items to execute in parallel and how the work-items are divided among work-groups.
- ▶ In **implicit model** programmer specifies only the total number of work-items to execute in parallel and the division into work-groups is managed by the OpenCL implementation.

OpenCL - Basics Overview

Soumyajit Dey, Assistant Professor, CSE, IIT Annapur


Now, in OpenCL, you have support for a hierarchical data parallel programming model, which would mean in the explicit model, the programmer defines a total number of work items to execute in parallel, and how the work items are divided among the work groups. And you can also have an implicit model where the program will only specify the total number of work items to be executed in parallel.

And the actual way in which the work items would be divided into the work groups will be managed by the OpenCL implementation. Now, this is important, this essentially means that as a programmer you can choose to explicitly specify the hierarchy of work items and while groups or you can actually let the runtime system take care of that, so, that would be the implicit model.

(Refer Slide Time: 17:29)

OpenCL Task Parallel Programming Model

- ▶ Logically equivalent to executing a kernel on a compute unit with a work-group containing a single work-item
- ▶ Users express parallelism by -
 - ▶ Using vector data types implemented by the device
 - ▶ Enqueuing multiple tasks



OpenCL - Runtime System Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

And this is logically equivalent to executing a kernel on a compute unit with a work group containing a single work item. So, this is more about the task parallel programming model. So, logical if same as like executing a sequential task and the parallelism is expressed by vector data type implemented by the device and you support in giving you can just also into multiple tasks.

(Refer Slide Time: 17:59)

OpenCL Program Structure

- ▶ Platform Layer API
 - ▶ Abstraction layer for diverse computational resources
 - ▶ Query, select and initialize compute devices
 - ▶ Create compute contexts and work-queues
- ▶ Runtime API
 - ▶ Launch compute kernel
 - ▶ Set kernel execution configuration
 - ▶ Manage scheduling, compute, and memory resources
 - ▶ Synchronization

Host program

- Query compute devices
- Create contexts
- Create memory objects associated to contexts
- Compile and create kernel program objects
- Issue commands to command-queue
- Synchronization of commands
- Clean up OpenCL resources


→ Platform Layer

→ Runtime

Kernels

- C code with some restrictions and extensions

→ Language



OpenCL - Runtime System Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, pleasure the introducing the name same will of course focus on this later on. Now, coming to this OpenCL program structures, like how really OpenCL program is ordered into different parts. But for that, we will first see that there are different supports in a typical OpenCL program. The first part will define what we call as the platform API. And the second part will deal with what we call the runtime API. So first of all, let us understand what is the platform API? Because then

we will see that how that is required to query the platform in the first day of the OpenCL force program.

So in OpenCL runtime system, you have a platform layer API, which is an abstraction layer for diverse computational resources and as a programmer. You have to use the query functions defined in this platform layer API for querying selecting and initializing different compute devices. And using this API, you can also create contexts and work use. So that is the high level API through which you have to actually create the different objects and obstructions like context, the memory objects, the program objects, and all that.

And also, once these, all these data structures are up and running, then you will be using functions from the runtime API to launch the kernels and manage the execution of the kernels, scheduled executions throughout the threads across the different devices. And utilize the memory resources and achieve synchronization as and when required across the threads and all that so if we just try and understand at a broad level, inside OpenCL program, the password would be to set up the data structures.

That are there, which will be managed by the platform API and then to do the actual workflow scheduling across the different devices which is managed by runtime API now, as we have been discussing in a OpenCL program, you will have initial part which is the host program part and also the kernel part. And the host program part we have a set of different steps to execute. And this is kind of a much more I would say, a bit more complex.

Then, normal CUDA was program where all you need to do is just define the basic variables and then define the setup the memory space for the GPUs and launch the kernels. The reasons are also something that we will discuss, but first let us understand what are the different steps? That OpenCL host program has to perform. The first thing it will do is it will query the different compute devices through the platform API and try to figure out the way the devices are arranged they are from which vendors.

And all that so, I mean, the query phase of course, I mean you if you know your executives and all that you can write an OpenCL code, which will not require to do this query and all that. But let us understand that we are trying to line we are trying to understand what is the generic way to write OpenCL programs, which can execute on any open source supported device. So for that, initially, the program that you write should be able to query the platform.

And get suitable values about its architecture information, which it will use for setting up the different memory objects and which it will also use to schedule the work items and all that. So using this query phase, you are first learning about the parameters of the underlying platform. And whatever you learn, let us say the number of devices from different vendors and all that that are useful for creating the context of execution as we have discussed earlier.

Inside the context, you will next be creating the memory objects to the context and then the next step will be to compile and create the kernel program objects. Now as you can see, this is also part of the runtime system. This is also supported by the runtime API. But these are all activities that are to be written in the first program itself. So now this seems a bit weird that we are saying, we have a program we have you we are we have first set up the context.

We have created the devices and set up the context and now we are seeing that, we will have further commands in the host program to create memory objects associated with context that is also fine. But the next thing is we will have compilation commands in the host program for combining the kernel code that is also going to come after those programs. Now, that is a weird thing because till now, we have been looking at things like we will write the code and then we will use a compiler to compile it but let us understand here.

The compilation process has to be managed by the host program and the way the program will be executed with will be managed by the runtime system. So the entire thing starting from I mean, starting from basic compilation to setting up the memory objects and using these compiled kernels to issue commands to the command queues, everything is performed by the host program. So this makes an OpenCL was struggling a bit more complex.

But at the same time, the structure of most of the OpenCL host programs are very similar. So once we are acquainted with one, writing another OpenCL host program is quite easy, because most of it is going to be a copy paste job. But, we need to understand what are the different steps so the takeaway at this point would be that the first program is not only responsible for declaring memory objects declaring inside the content, binding them to context.

But also for doing a runtime compilation of the different kernel program objects, and then, including the kernel launch commands in a specific order into the command queues. So, that also means the host program is also responsible for defining the command queues in a pod device manner. And in giving the different kinds of commands like data transfer commands, can launch commands synchronization commands all of them in the command queue.

And at the end of after that the next step will be launching the kernels through the runtime system. And once the kernels are finished their execution and the results have been properly computed to clean up the underlying resources, which were occupied. I mean, these are the underlying resources that the open source objects were occupying. So this entire orchestration of computation activity memory object definition activity command, including activity command execution, activity, synchronization activities, and the final cleanup.

This entire thing is managed by the OpenCL runtime system based on the query of the platform and the creation of context which is support supported by the platform layer. So these are the different phases through which a typical host program an open source program would go through. And so then the question is what is happening with the kernel the kernel is a simple CL program with certain restrictions and extensions.

And so, that would also be there about the host program, I mean, the point I was also trying to make multiple times again I would say that the host program will be also responsible for runtime compilation of this kernel problems.

(Refer Slide Time: 25:31)

Page 1/1

OpenCL and CUDA kernel Code

OpenCL

```

__kernel void vadd(__global float* a,
__global float* b, __global float*
c, const unsigned int n)
{
int i = get_global_id(0);
if(i < n)
c[i] = a[i] + b[i];
}



```

CUDA

```

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

```

OpenCL - Runtime Session Soumyajit Dey, Assistant Professor, IISc, IIT Kanpur

So, today will end with just a loop on a loop, and a simple OpenCL kernel, which would be our most popular vector at kernel. So on the right hand side, I have vector at kernel CUDA on the left hand side I have a vector at kernel in OpenCL. So as you can see, like we discussed earlier, the global keyword in CUDA gets replaced by underscore in 1 side only in the kernel keyword in OpenCL and here you had the input parameters and output buffer parameter.



And the size similar things, so, you have A, B and C, but they also come with this global keyword in case of the OpenCL, the OpenCL variables, if you go back to our earlier discussion.

(Refer Slide Time: 26:32)

Page 1/1

CUDA and OpenCL Correspondence

CUDA	OpenCL
__global__ function	__kernel function
__device__ function	no qualification needed
__constant__ variable	__constant variable
__device__ variable	__global variable
__shared__ variable	__local variable

OpenCL - Runtime Session Soumyajit Dey, Assistant Professor, IISc, IIT Kanpur

So, when we are talking about this, we had this global variable declaration here and that would mean that though they will be defined in that device's global memory, and the output will be computed again in the device each specific device's global memory. Now, coming back to the CUDA code you see that the first item was to compute a global thread id so this was all usual computation $i = \text{thread id} \times \text{block dim} + \text{block id}$.

And then based on the value of i being said the range, you are computing a normal C statement which is $C[i] = a + b \times i$ to here in OpenCL code, the last 2 lines are exactly the same, because fundamentally it is a C program with its own definitions of OpenCL types, data structures and runtime primitives. But look at the way i is computed, you simply have a call to something called a `get_global_id` function is providing you the global id for this thread.

The corresponding OpenCL work item and the work item will find will execute this edition for this specific value of i . So that is one good aspect here, you need not write this line time. And again, rather you just figure out what is the global id using this kind of function. So you have similar functions like `get_local_id` to get the id inside the thread inside the work group. And so the global id will give you the actual position inside the global arrangement of threads that we have defined.

And the 0 will signify that this is just in the first dimension that we are talking about and this is not a multi-dimensional kernel. Now, if we write 1 or 2 things like that will be we will actually talking about finding of the global id specifically in that dimension. So in case you are trying to linearize a multidimensional kernel, there are additional OpenCL, global kind of commands, which we will be discussing later on. So with this minor introduction to writing OpenCL kernels will end this lecture. And then next lecture we will look at the skeletal structure of OpenCL host program. Thank you for your attention will get maximum. Thank you.