GPU Architectures and Programming

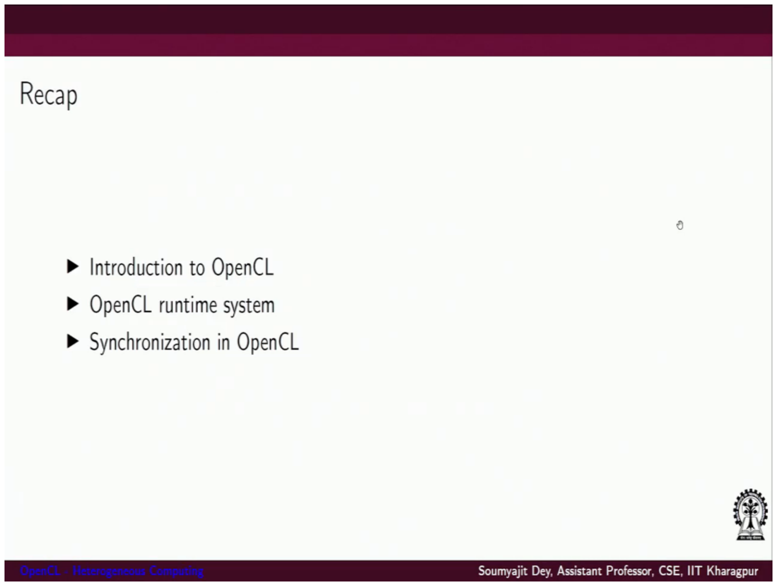
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture # 48
OpenCL – Heterogeneous Computing

Hi, welcome back to the lecture series on GPU architectures and programming. So I believe in the last week lecture, we have been talking a lot about open CL, as programming model and language, which is kind of runs parallel in compute capability of CUDA with some differences in terms of application platforms. So, this week, we will just continue a bit on using open CL for a heterogeneous computing environment.

Although I would say that, if I mean, although we will be talking about open CL and heterogeneous compile computing, but realize that CUDA is also the compute language which is used more often than not. We will also talk about how CUDA programs can be written in such a way that you can have multiple GPU devices on which the program runs given that some host CPU is orchestrating these different devices. So, starting with that motivation.

(Refer Slide Time: 01:23)



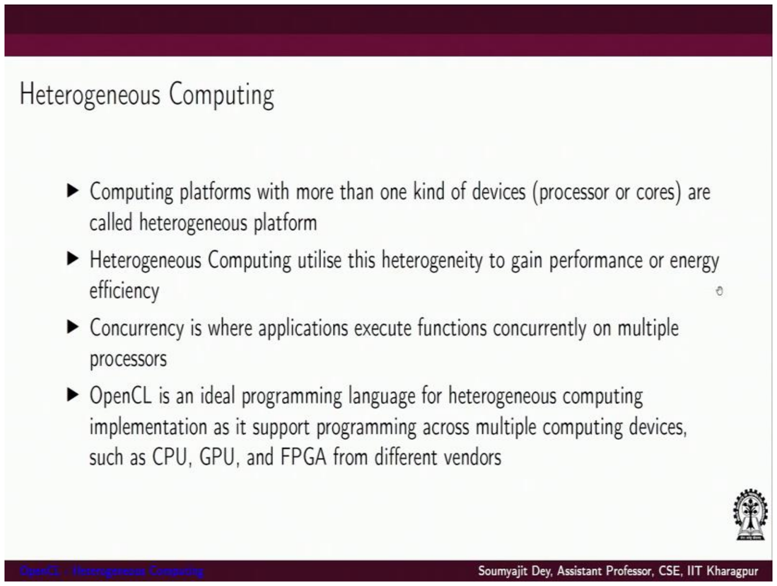
Recap

- Introduction to OpenCL
- OpenCL runtime system
- Synchronization in OpenCL

OpenCL – Heterogeneous Computing   Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, just want to say that in this case, if you want to recap, you can just look into the lectures we did earlier on introduction to open CL the details of the runtime system and the different synchronization primitives in open CL.

(Refer Slide Time: 01:40)



Heterogeneous Computing

- Computing platforms with more than one kind of devices (processor or cores) are called heterogeneous platform
- Heterogeneous Computing utilise this heterogeneity to gain performance or energy efficiency
- Concurrency is where applications execute functions concurrently on multiple processors
- OpenCL is an ideal programming language for heterogeneous computing implementation as it support programming across multiple computing devices, such as CPU, GPU, and FPGA from different vendors

OpenCL Heterogeneous Computing   Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

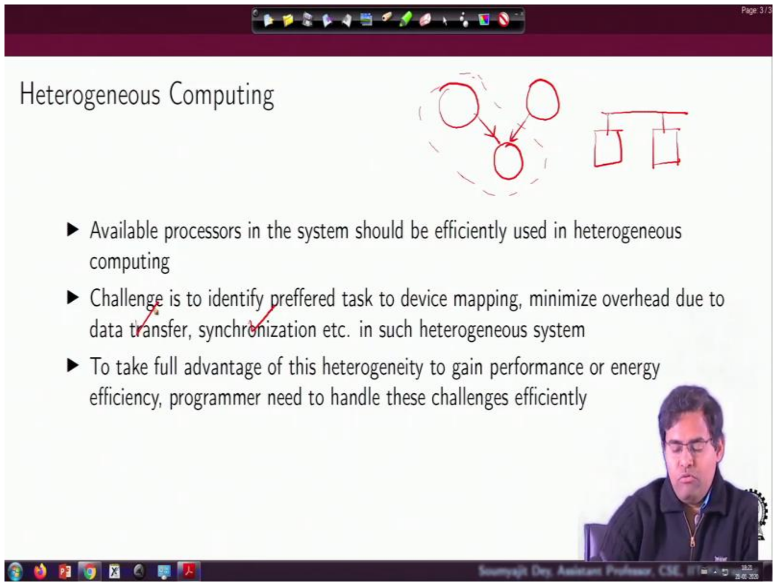
So coming to heterogeneous computing, what fundamentally is a trillion computing? So, essentially, we are talking about compute platforms which have got more than one kind of devices, it may mean processors are at a light level, we can talk about multiple different course, because I can have a compute platform with multiple different courts, and each court is kind of optimized to do different kinds of processing.

I can also think of a compute platform where on the PCI bus, I have different kinds of devices which are attached and they together are being also attached with a CPU device in the PC attached in the PCI bus and which is kind of orchestrating the execution of different kernels in these in this set of different devices. So, the idea of what radius computing is that how to make good use of this heterogeneity in terms of working platforms to gain in performance as well as energy efficiency.

So, if I have to do that, then I need to have multiple application kernels which are executing concurrently in these different devices. So for that, open CL is conceived as an idea programming language do because for because it supports programming across multi compute devices like CPUs, GPUs, as well as SPGs provided from different vendors of course, the vendor needs to support open CL, ICT loaders, essentially that the implementation of the origin urban specification that has to come from the vendors.

And there are a lot of vendors in recent times we have subscribed to this ideas. Just to let you know, in my in modern days, there has been an evolution in the Open CL community and there has been talked about a new programming paradigm called Falcon, which is kind of an open CL derivative popularly used to kind of develop data parallel applications in the context of android like mobile devices and I mean mobile devices, we subscribe to android like operating systems.

(Refer Slide Time: 03:45)



Heterogeneous Computing

- Available processors in the system should be efficiently used in heterogeneous computing
- Challenge is to identify preffered task to device mapping, minimize overhead due to data transfer, synchronization etc. in such heterogeneous system
- To take full advantage of this heterogeneity to gain performance or energy efficiency, programmer need to handle these challenges efficiently

So coming back to our ideas on heterogeneous computing, here we are talking about processors which are available in the system and we multiple of them are available and we want to use them efficiently. But of course, this brings in a lot of challenges. So if I want to characterize a problem here, let us consider the classical architecture to application mapping problem. So we have an application, which may comprise multiple data parallel kernels.

So if I am just trying to draw semantics of what we are trying to mean here, so let us say I have a set of data parallel kernels, which they are dependencies. So this is K 1, K 2, K 3 and like that K 4 this is the dependences. And let us say so this is my application. Let us say I have a target platform, we are on the bus, I have multiple processors, or let us open in terms of open CL, open CL devices, which are connected.

And let us say so they can be of different types. This can be a CPU or GPU type device, let us say, GPU device. And let us use a CPU device. And it is running the host program. And so, the objective is suppose we have this kind of a platform, we have got this kind of an application, how to map this application to this kind of a heterogeneous compute platform. So I call it a heterogeneous compute platform.

Because of this heterogeneity in terms of different devices that we have there can be much more variety. For example, it there can be let us say a device before, which is some neural computing in some other kind of accelerator card which is there which also understands how to execute an open CL kernel. So, that is the important thing we are subscribing to the idea. There is these are all different SMD compute platforms, they can be CPUs, GPUs, some other accelerator cards.

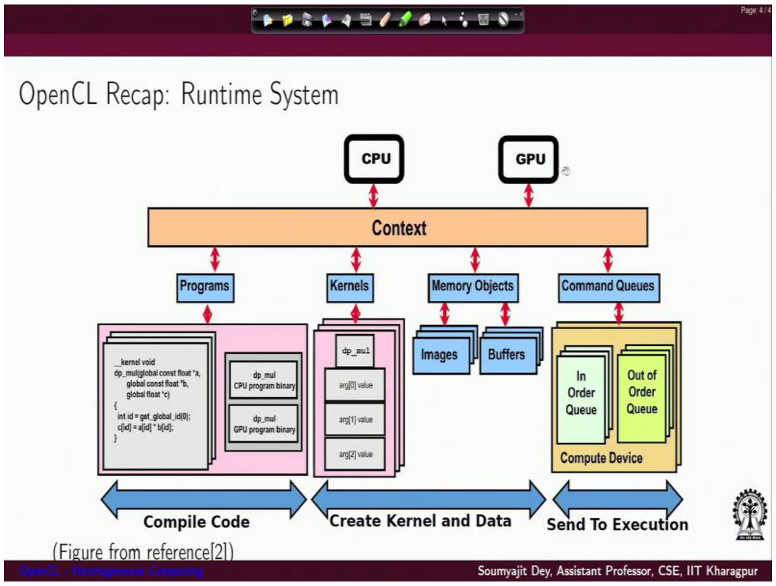
The constraint these all of them have onboard an implementation of the open source software specification. So that if this host program can discover this device, create a context for it or create a context with multiple instances of such devices and dispatchers kernels towards them, they will get executed. So, that would be what we call the classical application to architecture mapping problem restricted in the context of heterogeneous compute.

So, the challenge in this case is to identify preferred tasks to devise mapping, maximize, and minimize the overhead due to data transfer and synchronization in such a thread in your system. So that is also very important. When suppose, we consider again, let me just draw the first figure that if you have this kind of data panel kernels and you are trying to decide whether to map them in a single device or across multiple devices.

And the issue that may come suppose you decide on a mapping decision that you map 2 of these kernels in 1 device and the other countries and other device then you have a data transfer over it from this kernel to the rest of the kernels. But otherwise or also the execution here has to wait for the execution of this kernel to complete. So, you have a synchronization over it, you have additive transfer over it and based on how you do the mapping, you have an overall throughput.

So, depending on how you which aspect you want to optimize, you have to choose what kind of tasks will be phase mapping you want to have so, in order to take full advantage of this heterogeneity, and to gain performance in terms of performance or energy efficiency, the programmer will need to challenge handle these challenges very efficiently.

(Refer Slide Time: 08:20)



OpenCL Recap: Runtime System

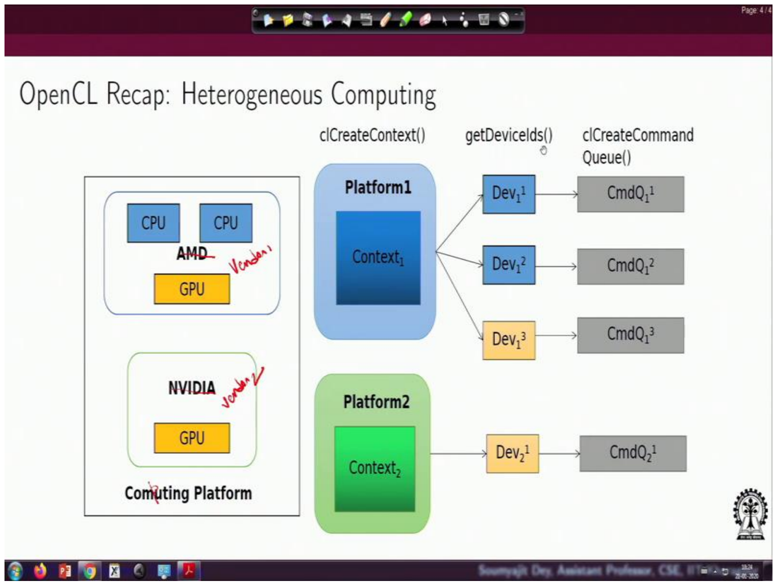
```
__kernel void
dp_mul(global const float *a,
       global const float *b,
       global float *c)
{
  int id = get_global_id(0);
  c[id] = a[id] * b[id];
}
```

(Figure from reference[2])

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, this is an overview of the open CL runtime system, which we discussed earlier from our last lecture. So, and that was that, I mean, that was actually trying to give you an overview of the different aspects that are really there, like you have to compile the code, you have to create kernels, and then you have to orchestrate the execution across devices and handle the transfer across devices. That is the overall job of this underlying runtime system.

(Refer Slide Time: 08:48)



Now, coming to this idea of heterogeneous compute, so, sorry for this, it will be a computing platform here and coming to this idea of heterogeneous compute, we are trying to give a very high level view here. So, consider that you have a platform with multiple vendor provided compute elements. Of course, they are connected and through the some share some shared memory or through based communication, they are communicate connected.

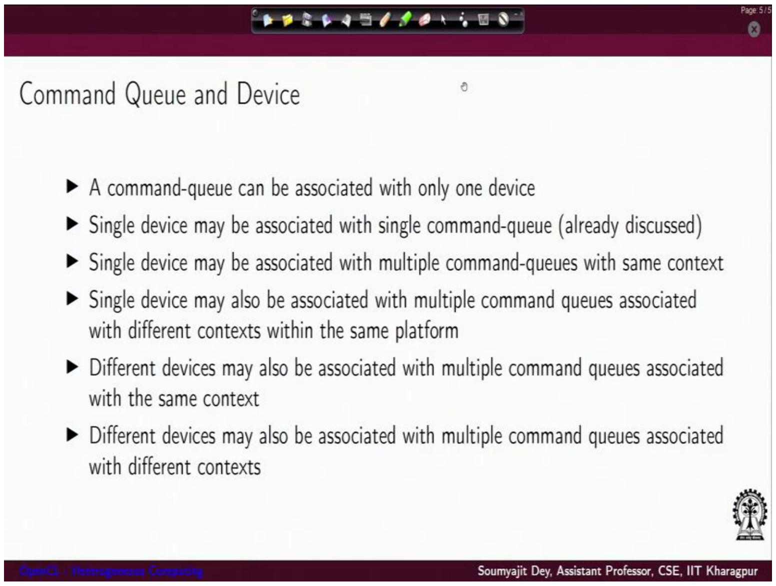
And you are using open CL to orchestrate the execution here. So, one possibility would be, for example, you can you are building we are just trying to give an example of how to build context like we have discussed earlier. So maybe I choose to create context for this platform from the vendors of AMD. We are just again taking examples here. And then we create another platform from the vendors of NVIDIA.

So, again, I will just repeat this our hypothetical is a very hypothetical example, we are thinking that such a computing platform is available is a totally theoretical thing. It does not actually mean of course, this kind of platform will never exist. So maybe we can just think that will we have things from different vendors. And we are creating, an open CL abstraction of these compute platform. Maybe we can create 1 context for platform 1, another context for a platform to, we go on to discover devices in each of the platforms.

So we will discover 3 devices here maybe we will discover one device here and P also go on to create command tools for each of them is like a very high level view of how things will look like we are trying to say that the because there are different kinds of compute elements which are present, the abstraction the definition of urban CLs support, things coming from different vendors, though, again, I will say that I do not think there are any example devices available where you have things from different vendors from the same these very hypothetical here.

So, I mean, it could have been just a vendor 1 and vendor 2. So, maybe we can say this is again from some vendor 1 and this is from some vendor 2 and you have created their corresponding devices. So, assuming that you have such a platform available, this is how you go about creating context discovering devices to the separate this commands that we have discussed and all that so on so forth.

(Refer Slide Time: 11:41)



Command Queue and Device

- A command-queue can be associated with only one device
- Single device may be associated with single command-queue (already discussed)
- Single device may be associated with multiple command-queues with same context
- Single device may also be associated with multiple command queues associated with different contexts within the same platform
- Different devices may also be associated with multiple command queues associated with the same context
- Different devices may also be associated with multiple command queues associated with different contexts

OpenCL - Heterogeneous Computing   Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, we know that there is this idea of command queues, which can be created in a pod device manner, every device should have its own command queue. So, once those things are set up, there are a few restrictions we need to get mine for example, A command queue can be associated with only 1 device. A single device may be associated with a single command queue, which we use a case we have already discussed.

However, it may be associated with multiple command queues also in the same context. So that can be another possibility. A single device may also be associated with multiple command queues, which are associated with different contexts within the same platform. Because if you remember we discussed earlier that even inside a platform I can create different contexts. So, maybe we can just go about and create 1 context.

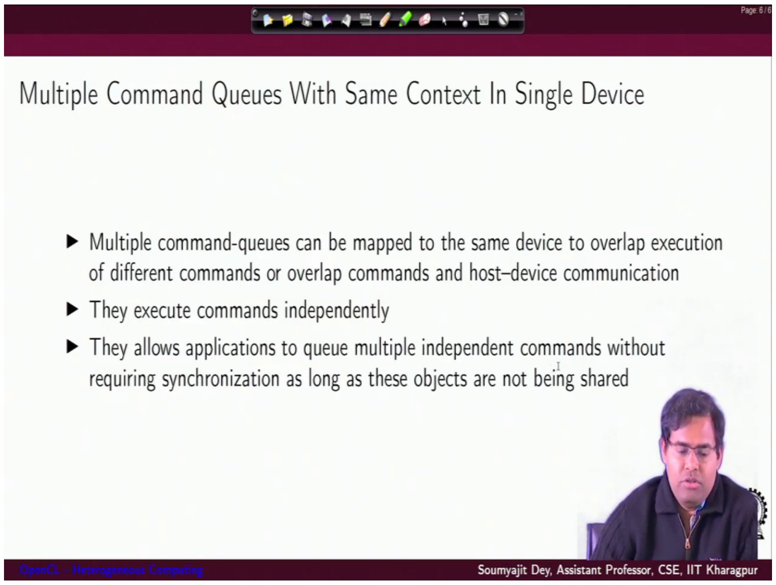
Using this set of devices, we can create another context again these set of devices, so, that would be 2 context here. So, in that case, I can have 2 different command queues, elevated with different contexts. Both of them are trying to access 1 device which in sitting in 1 of the context, again, these are all theoretical possibilities that opens your route support. For example, again, I will just repeat that you may not have this kind of a platform, but this is the abstraction that is supported.

And it also depends on whether the vendors themselves have their own even sell implementations. So, you can also have this issue that there are different devices associated with multiple command tools, which I will say that again with the same context. And different devices, which may also be associated with multiple command queues. And those commands queues stated with different contexts.

So, there is just understood the differences, you have command queues, you have devices and you have context. Every device is a command queue, every command queue should subscribe to some device. We are trying to say what are the different options here, I can have multiple commands queue for a single device, I can have multiple command queue bringing to coming from different contexts mapping to some device.

They are giving conditioning commands to some device in contains 1 context, I can also have in the same context multiple command queues. And I can have these multiple command queues with different devices. And I can also have the same thing from the point of view of different contexts.

(Refer Slide Time: 14:28)



Multiple Command Queues With Same Context In Single Device

- Multiple command-queues can be mapped to the same device to overlap execution of different commands or overlap commands and host–device communication
- They execute commands independently
- They allows applications to queue multiple independent commands without requiring synchronization as long as these objects are not being shared

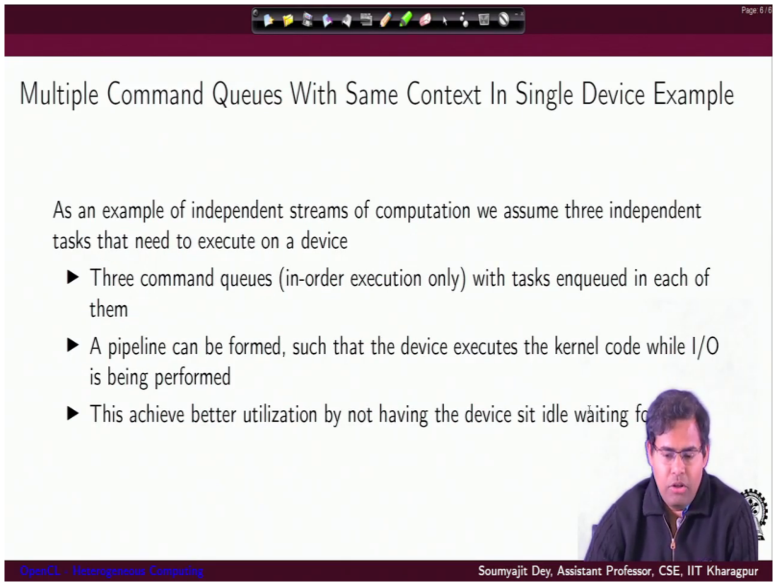
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So let us take these cases one by one. So, considered that I have multiple command queues with same context in a single device. So essentially, we are saying that multiple command queues can be mapped to the same device. But why? The fundamental reason would be that where the device is getting commands which are issued from the command queues, now, inside recently, the command queues so you are issuing commands in order. If you want to overlap the execution from the different common tools, then you may have multiple common tools mapped to the same device.

So that let us say 1 command queue is in queuing, from 1 command queue has a read or write operation and another command queue as a kernel launch operation and they are overlapping on the device that is a possibility. So, essentially what happens in this case, they are executing commands independently. Each of the command queues are throwing up their own commands to the device that those commands are executing independently.

I mean, among command, I mean, there is an independence among commands coming from different command tools. They allow applications to queue multiple independent commands without recording synchronization as long as the objects are not shared. So there is also an advantage that if you have such multiple command queues, you are able to independent commands without requiring synchronization why because the moment you have commands in the queue, you have some ordering already being present.

(Refer Slide Time: 16:12)



Multiple Command Queues With Same Context In Single Device Example

As an example of independent streams of computation we assume three independent tasks that need to execute on a device

- Three command queues (in-order execution only) with tasks enqueued in each of them
- A pipeline can be formed, such that the device executes the kernel code while I/O is being performed
- This achieve better utilization by not having the device sit idle waiting fo

OpenCL - Heterogeneous Computing

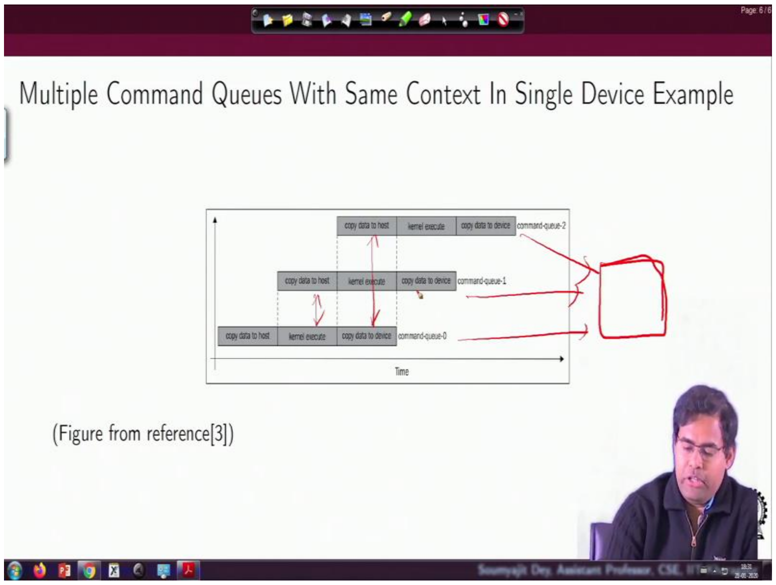
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, consider this case you have multiple command queues with same context in single device as an example of that, you can have independent streams of computation and you assume that three independent tasks are there which need to execute on a device. So, we are saying that you have 1 device and I want a device available for executing 3 tasks concurrently in an independent way. So, I will just set up 3 command queues in order execution with tasks include in each of them that would form a pipeline.

So, that each device executes the kernel code file the I/O is transferred. So, let us say from 1 command queue I am issuing an I/O command like in read or write command there are only from the other command queue, I am issuing a kernel launch command, since the device will have separate hardware for read/write and execution using the streaming multiprocessors these operations can overlap they can be done concurrently.

So this will help in achieving that vision by not having the device sit idle and waiting for data, because while it is the copying doing the reader read or write operations, I have the device executing the kernel, some other kernel coming from one of the command queues.

(Refer Slide Time: 17:34)



Multiple Command Queues With Same Context In Single Device Example

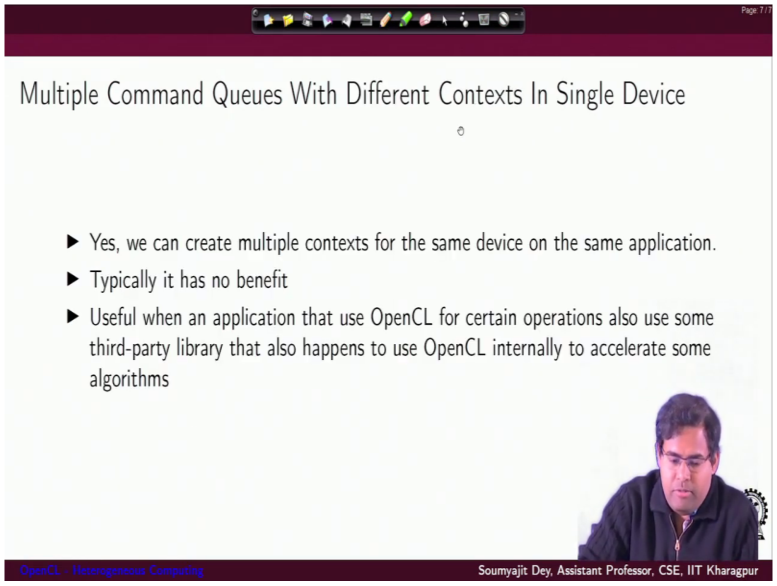
(Figure from reference[3])

So if I have this kind of multiple command queues within the same context, and they are issuing commands to a single device, this is how it helps me to overlap executions is just like a pipelining with speed up our movement in computer architecture. So let us say a command queue 0 is providing this command sequence copy data to device kernel execute copyright to host and there is a simple sequence for another kernel. And those commands are coming from command queue 1.

Similarly, I have such a sequence from command queue 2 when I have this kind of execution and this execution sequence is going to the same device. They are going to the same device sitting in the same context, then I can see that this data and execution can happen in parallel copy that copy data to host copyright to device and execution can also happen in parallel provided you have got multiple copying engines, which are often available.

So one copying in takes of copying from the device of the host and other copying in this case of copying from the data from the host to the device. So in that way also I can have this kind of overlap execution. So that is the advantage in Open CL of having multiple command queues with same context in a single device.

(Refer Slide Time: 18:54)



Multiple Command Queues With Different Contexts In Single Device

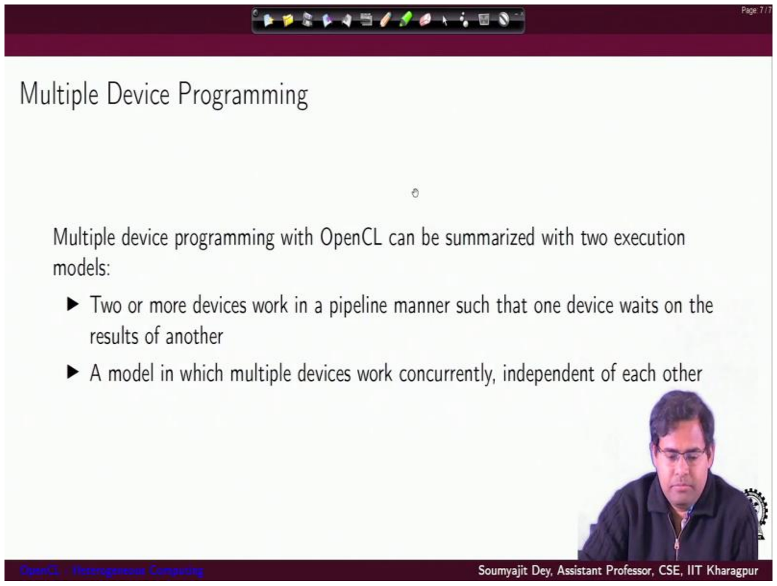
- Yes, we can create multiple contexts for the same device on the same application.
- Typically it has no benefit
- Useful when an application that use OpenCL for certain operations also use some third-party library that also happens to use OpenCL internally to accelerate some algorithms

OpenCL : Heterogeneous Computing Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So in the mean, we can create such multiple context for same device on the same application. And but what if I have multiple contexts? Is there a benefit? Typically? No, because it does not. So this is another case where I have such multiple command queues but the command queues are coming from different contexts. So the kind of over left execution advantage I get inside the context, that would not happen now? Typically, that is why there is no benefit.

It is useful when an application that uses open CL for certain operations will also use some third party library. That also happens to use open CL internally to accelerate some algorithms. So that so you have one context for the application. You can have, you can create a separate contest where you load that third party library. We use it open CL internally for an algorithmic acceleration. So those are the cases where for providing support, you can land up in this kind of a setting.

(Refer Slide Time: 20:11)


Multiple Device Programming

Multiple device programming with OpenCL can be summarized with two execution models:

- Two or more devices work in a pipeline manner such that one device waits on the results of another
- A model in which multiple devices work concurrently, independent of each other

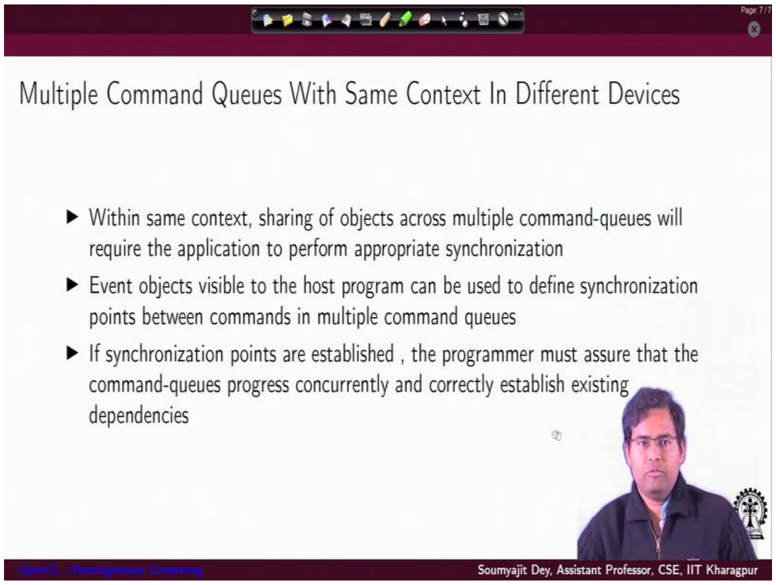
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, coming to this device, the programming of multiple devices. So this is all about single device, multiple command queues, are sitting in the same context and different context. Now let us talk about multiple devices. So, for that, I will have 2 execution models, I may have 2 or more devices, which work in a pipeline manner, such that 1 device waits on the results of another alternative like and have a model where the multiple devices work concurrently.

They are independent of each other. So, there does not consider different devices, which are in the same context and we have multiple concurrent command queues for these different devices. So for multiple devices in a system, for example, a CPU and the GPU or multiple GPUs, each device will need each device will need its own command queue? So you have a context where you have multiple devices, and each device will need its own command queue.

A standard way to work with multiple devices on the same platform is creating a single context as memory objects are globally visible to all devices within the same context. And an event is only valid in a context where it is created. So just to remember that if you have multiple devices, you may want to create this kind of a single context. Because then the memory objects which you create before actual kernel execution, they are global inside the context, they can be accessed by all the devices. And same thing will hold for events that you define.

(Refer Slide Time: 21:51)


Multiple Command Queues With Same Context In Different Devices

- Within same context, sharing of objects across multiple command-queues will require the application to perform appropriate synchronization
- Event objects visible to the host program can be used to define synchronization points between commands in multiple command queues
- If synchronization points are established , the programmer must assure that the command-queues progress concurrently and correctly establish existing dependencies

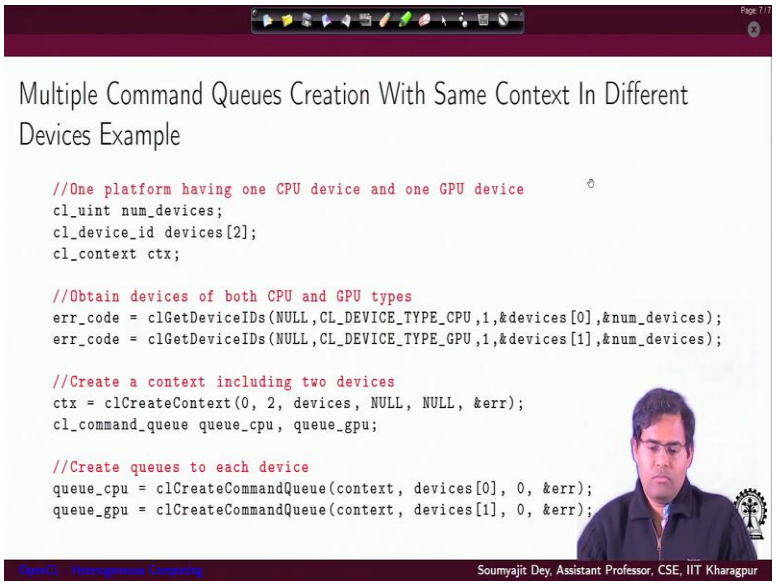
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, inside the same context, you can also have this sharing of objects that you define across multiple command queues, but it will record the application to perform appropriate synchronization. So that dependencies are preserved. Now, event objects that are visible to the host program, they can be used to define synchronization points between commands in multiple commanders. I like we have seen some synchronization examples earlier. But those were for a same command queue. We are just trying to say that since this synchronization events there I mean the event lists in open CL.

They are global data structures inside the same context they are actually available across command queues. So I can have events in events that have been triggered in 1 common queue, they can actually be used to trigger events in another command queue. So, if the synchronization points are established, then the programmer must ensure that command queues progress concurrently and establish the current dependencies of course like we have seen earlier, that using the event model.

That event lists and triggering events using this programming model to establish existing dependencies is something that is on the hand of the programmer and the programmer has to ensure that it really happens nicely.

(Refer Slide Time: 23:14)



Multiple Command Queues Creation With Same Context In Different Devices Example

```
//One platform having one CPU device and one GPU device
cl_uint num_devices;
cl_device_id devices[2];
cl_context ctx;

//Obtain devices of both CPU and GPU types
err_code = clGetDeviceIDs(NULL,CL_DEVICE_TYPE_CPU,1,&devices[0],&num_devices);
err_code = clGetDeviceIDs(NULL,CL_DEVICE_TYPE_GPU,1,&devices[1],&num_devices);

//Create a context including two devices
ctx = clCreateContext(0, 2, devices, NULL, NULL, &err);
cl_command_queue queue_cpu, queue_gpu;

//Create queues to each device
queue_cpu = clCreateCommandQueue(context, devices[0], 0, &err);
queue_gpu = clCreateCommandQueue(context, devices[1], 0, &err);
```

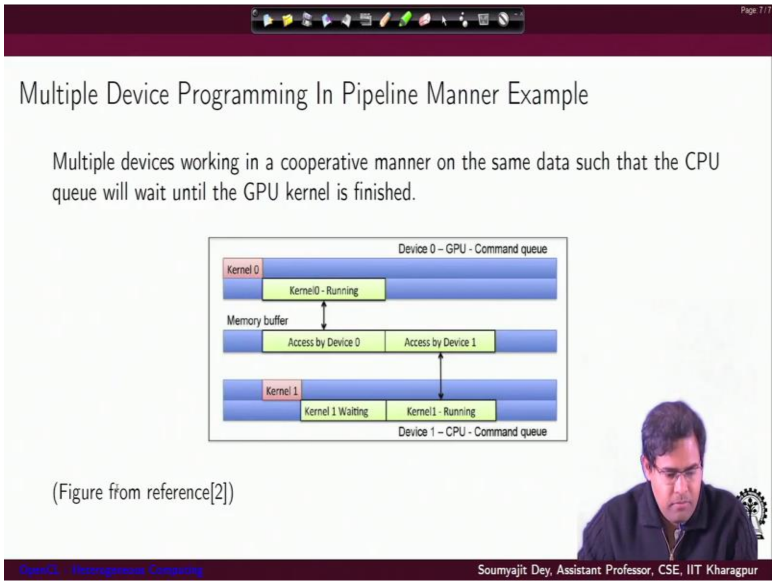
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, consider some examples now for example, multiple common queues which are created with same context in different devices. So, you have 1 platform having 1 CPU device and 1 GPU device. So you have these devices area so what do you do I mean, by just giving an example of a similar code. First, the difference only will be creating multiple command queues for multiple devices. So first to discover the devices using this CL get device Ids.

So you discovered the CPU type device and the GPU type device. So and then, we are assuming that we already know that in there we have this 1 CPU device and 1 GPU device. Then you click context where you use both these 2 devices. So there is a single context that you create for 2 devices? And then inside this context so then inside this context, you will be creating cues for each of these devices. And that is done in the next 2 commands. So you have CL create command queue here for creating the command queues for the same context.

And then so, but 1 of them is for the GPU. So as you can see that for the same context, you have, the 2 devices use the GPU device. So for that you have queue GPU and similarly for queue CPU.

(Refer Slide Time: 24:52)



Multiple Device Programming In Pipeline Manner Example

Multiple devices working in a cooperative manner on the same data such that the CPU queue will wait until the GPU kernel is finished.

(Figure from reference[2])

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

And then it will be possible using this programming model that for these 2 devices, you can have overlapped execution that you should be able to launch 2 kernels. So that 1 kernel is executing in the GPU device. The other kernel is executing in the CPU device, but they are sharing data through a memory buffer. So, it will help us to the point that the multiple devices will be working in a cooperative manner on the same data says that the CPU will wait until the GPU kernel is finished.

So as you can see from the dependency diagram, you have kernel 0 in the command queue. Kernel 0 goes from dispatch to kernel 0 starts running after retains. So the consider timing here in this axis. So you in the memory buffer, it is accessed and this is the data is written after the data is written. You may have the data being accessed by the device 1 where the kernel was waiting and it will start running.

So we know how to write such quotes. And we will, go into I mean looking into such programming models in a deeper way. And maybe we will end this lecture by stopping at this point of time. And that would be all thank you.