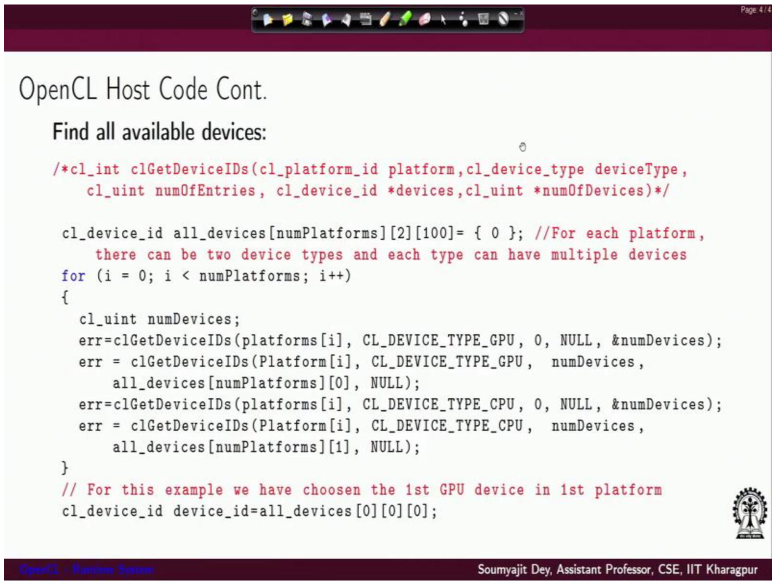
GPU Architectures and Programming
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture # 44
OpenCL - Runtime System (Contd.)

(Refer Slide Time: 00:33)



Page 4 / 4

OpenCL Host Code Cont.

Find all available devices:

```
/*cl_int clGetDeviceIDs(cl_platform_id platform,cl_device_type deviceType,
   cl_uint numOfEntries, cl_device_id *devices,cl_uint *numOfDevices)*/

 cl_device_id all_devices[numPlatforms][2][100]= { 0 }; //For each platform,
    there can be two device types and each type can have multiple devices
 for (i = 0; i < numPlatforms; i++)
 {
   cl_uint numDevices;
   err=clGetDeviceIDs(platforms[i], CL_DEVICE_TYPE_GPU, 0, NULL, &numDevices);
   err = clGetDeviceIDs(Platform[i], CL_DEVICE_TYPE_GPU,  numDevices,
       all_devices[numPlatforms][0], NULL);
   err=clGetDeviceIDs(platforms[i], CL_DEVICE_TYPE_CPU, 0, NULL, &numDevices);
   err = clGetDeviceIDs(Platform[i], CL_DEVICE_TYPE_CPU,  numDevices,
       all_devices[numPlatforms][1], NULL);
 }
 // For this example we have choosen the 1st GPU device in 1st platform
 cl_device_id device_id=all_devices[0][0][0];
```

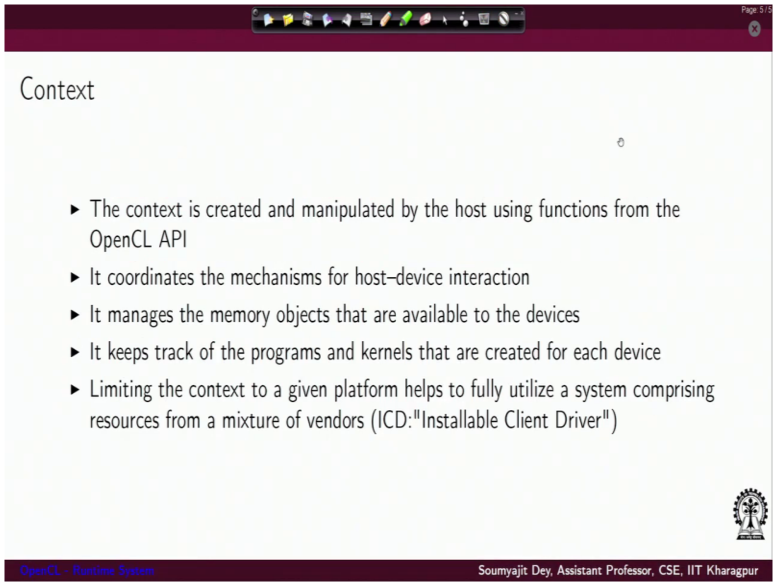
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Hi, welcome back to the lecture series on GPU architectures and programming. So, I believe in the last lecture we have been discussing in detail about simple OpenCL host program, which is actually looking into the architecture and discovering the different devices. And let us try and figure out now how after discovering the devices things are going to work. So the way will be interspersing. The presentation is that before going into any segment of the code will be again kind of doing a small recap of the concepts.

We have discussed earlier, but we will do it in a more formal way, the properties of those concepts and why they are useful. And then we will be showing the corresponding program segments there so we will structure. So if you recall from our last lecture, here, what we did was, we have discovered the different devices, we are storing the GPU type devices in the in for each platform. In the first part of the day, we are storing the GPU type devices, ids.

And then the second part of the id we have the CPU type device ids. So for plot platform 0, the first device, the like in all devices, as you can see all devices 000. In this, the first device that will you have is going to be a GPU device. So let us say we get the device id here in our device type in a device id variable of type CL device id.

(Refer Slide Time: 02:01)



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, CSE, IIT Kharagpur

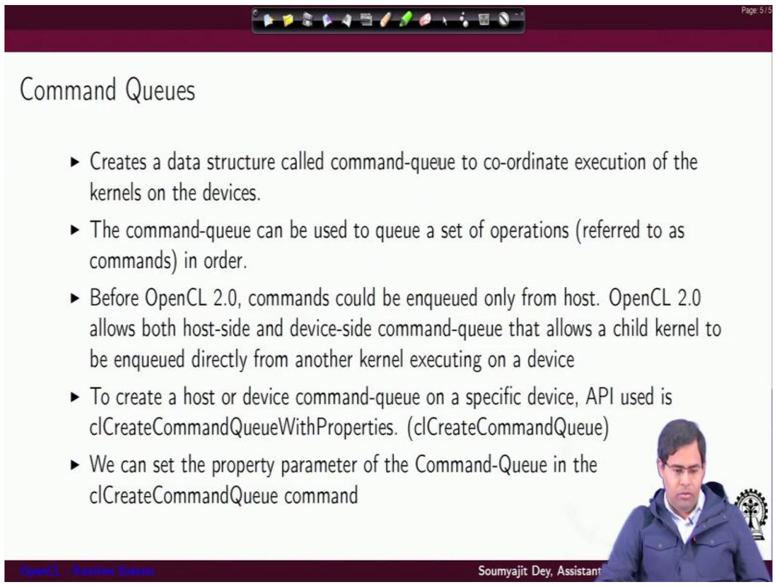
Now, let us see how this can be used to do some real work. So, if you remember from our earlier lecture, so we discussed setting up a context and then building up a context using devices that you slowly keep on discovering in the system and all that so now we are going to really do that using OpenCL code. So what is really important is to remember a context is created and manipulated by the host using functions from OpenCL API.

So it is basically an OpenCL host program, which is defining the context using the platforms and devices. And then inside the context, the host program will perform interactions with the device inside the context. It is the memory objects are also managed. And because you can manage you can share data among memory objects sitting inside 1 context because that is like an abstraction. So for all available devices inside the same context, the management of memory objects will be happening inside that context itself.

It also keeps track of the programs and kernels that are created for each device. So whatever devices you have created for which, what are the programs and kernels that are there, they are all managed by that context? Now, typically, that rule is that you should limit a kernel to a given platform. Because then you can fully utilize the system comprising resources from a mixture of vendors, because x is render will provide you an ICD or by installable client driver opens your implementation of that vendor.

So that is the lower level library, which will be running and managing the OpenCL code in that vendors platform. So if you need to limit the context, specific to vendor specific platforms, and inside the context, you can have multiple devices working together. Which are also part of that vendor distribution.

(Refer Slide Time: 04:08)



Command Queues

- Creates a data structure called command-queue to co-ordinate execution of the kernels on the devices.
- The command-queue can be used to queue a set of operations (referred to as commands) in order.
- Before OpenCL 2.0, commands could be enqueued only from host. OpenCL 2.0 allows both host-side and device-side command-queue that allows a child kernel to be enqueued directly from another kernel executing on a device
- To create a host or device command-queue on a specific device, API used is clCreateCommandQueueWithProperties. (clCreateCommandQueue)
- We can set the property parameter of the Command-Queue in the clCreateCommandQueue command

Soumyajit Dey, Assistant

Now, if you remember earlier we discussed the idea of command queues like why do we require a queue here so, that you can give an ordering of the commands and all that so, that is also from a programmers point of view that is also a specific kind of data structure that would be you need to define for coordinating the execution of the kernels. Now, inside the command queue, you can you can actually specify what are the queuing or what are the queuing actions to be taken which are the commands to be executed in which order and all that.

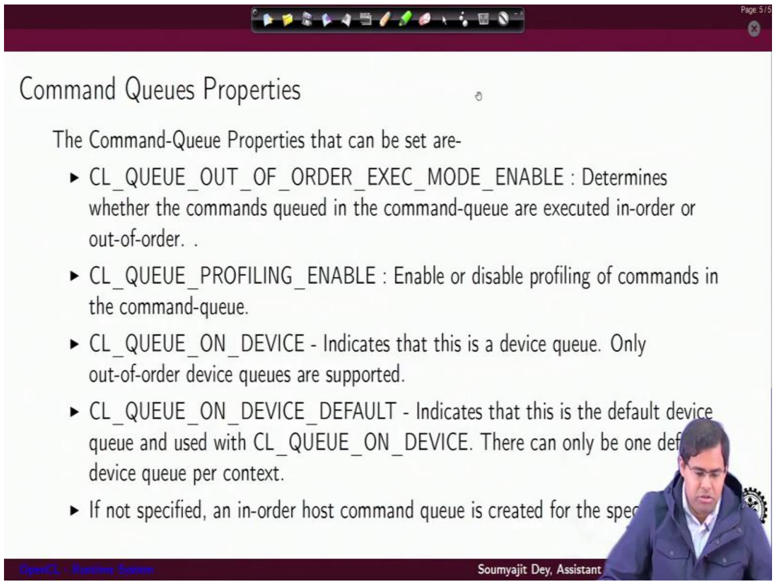
Now, before OpenCL 2.0 commands will be included only from the host that means only the host can specify which kind of read writer kernel execution operation is to be done, but from the OpenCL 2.0 specification both hosts side and device side command use these concepts have come up like a code that is executing in a device like a device code like a kernel is executing in a OpenCL device from that itself, I can create child kernels, which can include directly.

So, will we like to summarize like the device side command queue, you can have a host side command queue, you can also have a device set command queue. And this allows that you can spawn a child kernel from an execute from a kernel which is already executing on a device and, and from this child and you can allow the child kernel to be enqueued directly from the currently executing kernel.

So this is essentially known as device side enqueuing, maybe we will touch upon it later on now let us go straight to the simple basics here. So in order to any of these commands use either a host side command queue and a device command queue on some specific device, there is a specific API that is used called clcreate command queue with properties. So, first of all let us understand you need to operate in a part you can have command queue to define that many levels, you can have command queue in a pod device manner.

You can have multiple command queues giving, commands to a device. So in what are the possibilities? We will see all those now, you can see we can also set the property parameter of the command queue in the CL create command queue command like what kind of executions will be supported and all that.

(Refer Slide Time: 06:47)


Command Queues Properties

The Command-Queue Properties that can be set are-

- CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE : Determines whether the commands queued in the command-queue are executed in-order or out-of-order. .
- CL_QUEUE_PROFILING_ENABLE : Enable or disable profiling of commands in the command-queue.
- CL_QUEUE_ON_DEVICE - Indicates that this is a device queue. Only out-of-order device queues are supported.
- CL_QUEUE_ON_DEVICE_DEFAULT - Indicates that this is the default device queue and used with CL_QUEUE_ON_DEVICE. There can only be one def device queue per context.
- If not specified, an in-order host command queue is created for the spe

Soumyajit Dey, Assistant

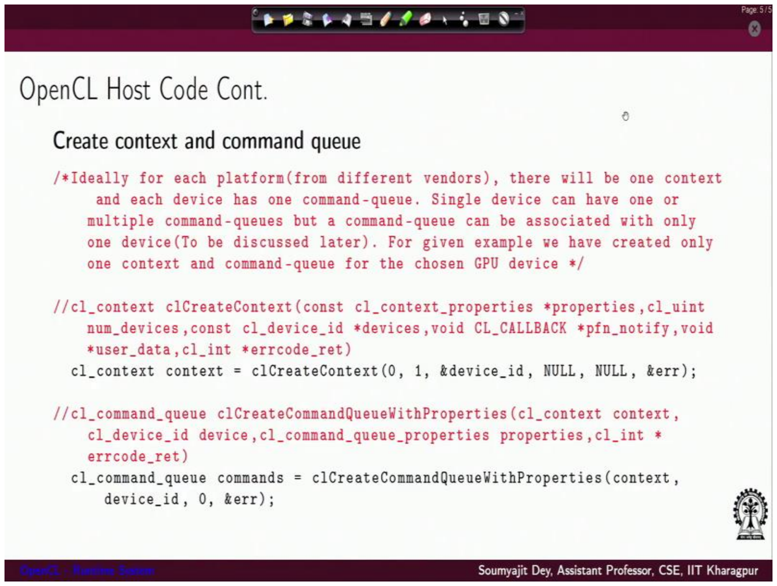
So this is just like a basic introduction to the structure called command queues. Now, what are the properties that are supported? For example, you can have a property enable, which is CL queue out of order execution mode enable, if you are setting this to be true that determines that the commands in the command you can execute out of order, I mean. So, it will be executing out of order. Also you will be in you will be interested in doing some runtime profiling like what amount of time exactly is taken by some read/write or execute kind of operation in the on the device.

So, you can enable the profiling by using this flag CL queue profiling enable to indicate that some command queue is a device specific you, you can have this flag enabled. So only in you in such device queues, there is something important will feel for your facilitator. They are only out of order queues are possible in devices. Now, this other property CL queue on device default, which indicates that this is the default device queue and to be used with CL queue on device.

We will see some examples now there can be only 1 default device queue per context. So inside a context for the device, you can have 1 default device queue? So these are all again, things specific to device queues like in a device, you can have a queue with out of order enable for our device, you can have only 1 default queue like that now, if it is not specified, and in order host command queue is created for the specific device, nothing else is specified, you have created a command queue for a specific device.

That is the host side command queue, which is going to execute commands in the execution in the order in which they have been queued not in out of order way. So there is a default that if you do not specify something else, for a specific device, you have a host site in order command queue.

(Refer Slide Time: 08:53)



OpenCL Host Code Cont.

Create context and command queue

```
/*Ideally for each platform(from different vendors), there will be one context
     and each device has one command-queue. Single device can have one or
    multiple command-queues but a command-queue can be associated with only
    one device(To be discussed later). For given example we have created only
    one context and command-queue for the chosen GPU device */

//cl_context clCreateContext(const cl_context_properties *properties,cl_uint
    num_devices,const cl_device_id *devices,void CL_CALLBACK *pfn_notify,void
    *user_data,cl_int *errcode_ret)
  cl_context context = clCreateContext(0, 1, &device_id, NULL, NULL, &err);

//cl_command_queue clCreateCommandQueueWithProperties(cl_context context,
    cl_device_id device,cl_command_queue_properties properties,cl_int *
    errcode_ret)
  cl_command_queue commands = clCreateCommandQueueWithProperties(context,
     device_id, 0, &err);
```

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So first of all, we will just see such a simple examples. So you create context and command you how does that work? So, ideally for each platform which comes from different vendors, there will be 1 context and each device will have 1 command queue, for each platform there is important from different vendors from each platform you have 1 context and each device will have 1 command queue. So, using that command queue give to commands read/write execute kind of commands to the device, single device can have one or multiple command queue.

Now that is also important you can have multiple command queue for the same device. But what is not possible is a command queue can be associated with only 1 device I cannot have a command queue which is giving which is churning out command to execute in multiple devices will discuss this simulator. So for this example, we have created only 1 context will create on 1 context and will create 1 command queue for the single GPU device which we choose write here this 1 so, here as you can see, for the 0 platform.

We have chosen the fastest GPU device and afford that GPU device will be doing all these definitions of context queue and all that so that I can create a contest pack and tag the GPU inside the contest create a queue, which will be enqueuing commands targeting that device and all that, so 1 thing 1 must appreciate, while we go through these concepts is how genetic this system is I will just repeat that you should not get overwhelmed with such a lot of definitions.

Let us understand that this has been kept like this to make it generic. You can execute so many different kinds of devices coming up I mean, if they are you have multiple devices arrange together you can so this gives you a way to organize their architecture, scheduled programs on them and create dependencies and all that now, of course, the other important point would be while this code looks so difficult most of these are kind of very means going to be repetitive in the context of program execution.

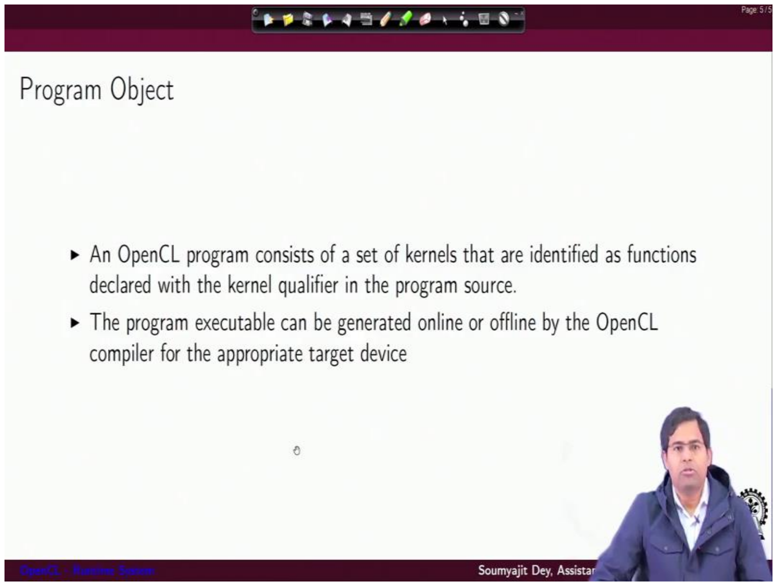
So, those can just be copied if you have 1 implementation available with you. So, the first thing that would come is how to create a context we have been talking about context. So this is a call for that clcreate context is going to return you can OpenCL context and it can go through several parameters, which are specified here. Now, initially, when you are just creating, the context, we are just, we will just discuss that for we just specify what are the devices to be attached with in this case we are just given 1 device.

So we just have a single pointer to that devices id if I have a collection of devices, this would essentially be pointers for that so I have just created a context with 1 GPU device. In general, as I am repeating, I can create a context with multiple devices, multiple CPUs and GPUs, kinds of devices and all that by creating an app device id array and passing it here the next thing that comes is I have to create the command queue.

So, this would be the command for that CL create command queue with properties. So, you are creating a command queue for this context, which has been provided. And since we have discussed so that are inside our context, I can have multiple devices. For each device, I need to have at least 1 command queue, which will issue commands for the device. So will, create now that command queue for the context.

And also for some specific device in the context I am trying to impress upon why we should also have the device id here. Because although here I am creating a context with a single device. I could have created a context in general with multiple devices. So then when I am creating command queues, I need to specify for which contexts which device I am creating the queue.

(Refer Slide Time: 13:09)



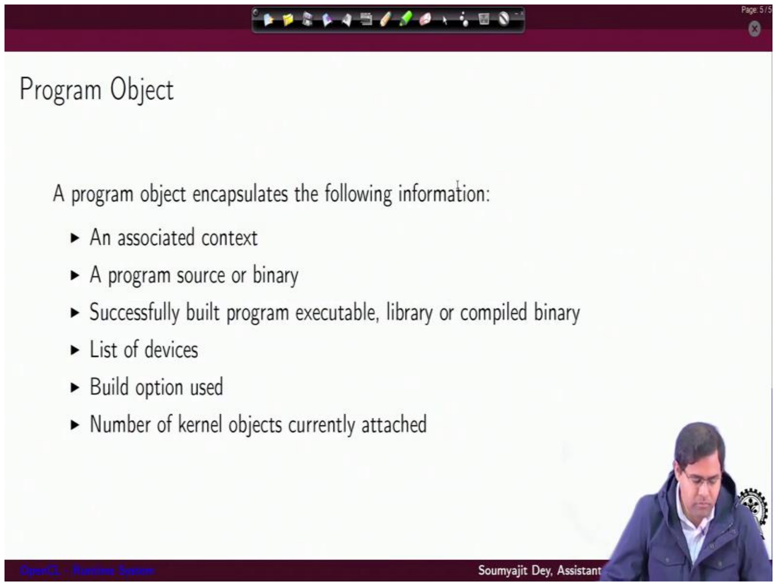
Program Object

- An OpenCL program consists of a set of kernels that are identified as functions declared with the kernel qualifier in the program source.
- The program executable can be generated online or offline by the OpenCL compiler for the appropriate target device

So, the next thing would be a program object. So an OpenCL program consists of a set of kernels that are identified as functions declared with the kernel qualified in the program source. So, it will write the OpenCL kernel with this qualified, kernel in this in before the declaration, as you can see in our OpenCL kernels example now, the program executable can be generated online or offline by the OpenCL compiler for appropriate target device as you can see, 1 option is I like I said earlier, the kernel can be specified inside in a way that finally the host program would view the kernel as a character as a stream.

So it will look at it like that and it will compile it and then create the program object and then create commands around it. Now, this idea of creating the program object, we are saying that it can be generated either online or offline by the OpenCL compiler for a suitable device. In this case, we just saw as many the host code where the kernel would be the core from the kernel source code. Creation of the program executable will be done online during the execution of the program.

(Refer Slide Time: 14:31)



Program Object

A program object encapsulates the following information:

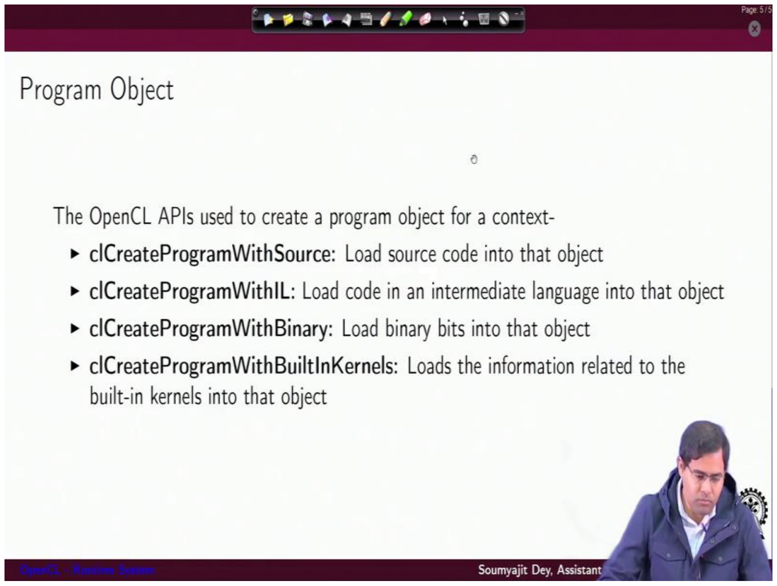
- An associated context
- A program source or binary
- Successfully built program executable, library or compiled binary
- List of devices
- Build option used
- Number of kernel objects currently attached

Soumyajit Dey, Assistant

So, what I mean what essentially is there in that program object so, it is less like an encapsulation of the following information like it would necessarily need and context that the program object is also aided with this context. That means it can access the data offers and devices, I mean, on that context. It can be enqueued in the command queues which are on that context like that it will also have a binary approach to execute and the program engineer, also the successfully built program executable a library or a compile library.

And it contains the list of devices on which as I am saying that can execute, what are the build options for the program object. Now, we are not discussing that too far. You can look at it later on and what are the number of kernel objects that are attached so you can have multiple such kernel objects for the problem object.

(Refer Slide Time: 15:28)



Program Object

The OpenCL APIs used to create a program object for a context-

- **clCreateProgramWithSource:** Load source code into that object
- **clCreateProgramWithIL:** Load code in an intermediate language into that object
- **clCreateProgramWithBinary:** Load binary bits into that object
- **clCreateProgramWithBuiltInKernels:** Loads the information related to the built-in kernels into that object

Soumyajit Dey, Assistant

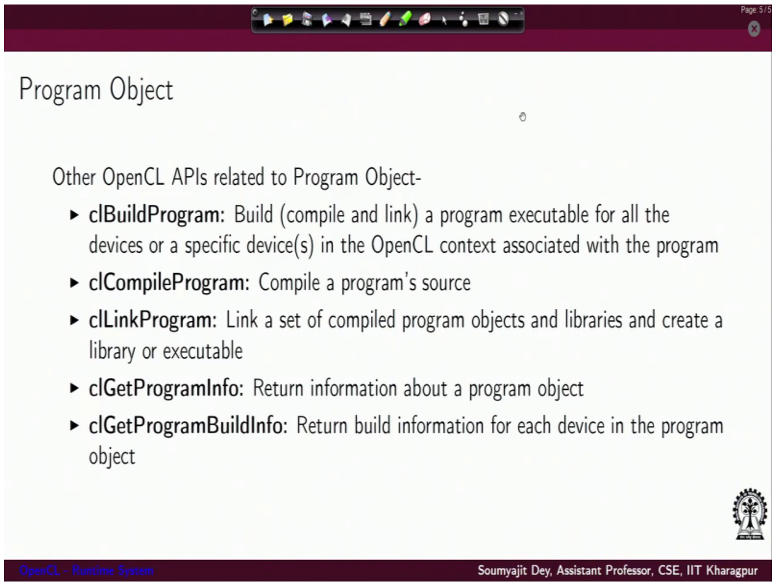
Now, for creating such a program object for a specific context, there are some functions which need to be supported by OpenCL API. And they are essentially this CL create program with source the function of this would be that it loads the source code into that object. You have defined that object and you are loading the source code there clcreate program with IL so load the code the intermediate language into that object again we are not going into the details of this.

Now and next thing is CL create program with binary you load the binary bits into objects. So essentially depends on in what form the program is available to you is it available source. Like I said a string, we will see some examples, or is it available in the form of an intermediately compiled language like from the source you have made some compilation to in kernels of representation. And from that you want to create the program.

For the program focus, we want to create a program object or even the binary is available, and you want to just create them from CL program binary. So, as I am said, his id of OpenCL online or offline execution would mean that you may perform runtime compilation, you may perform runtime intermediate cogeneration or you may have the binary available. For all those 3 possible options. Your host program will actually use 1 of these options to compile the program to actually create the program object using the format.

In which the program is available whether from source from intermediate level code or from the binary for creating the program object and then you have CL build program create program with built in kernels. So, this loads the information related to build in kernels you may have kernels which are already built in to load information about them because for those are already available to them. So, the program object and does not need to create those kernel objects but obtain the information about them.

(Refer Slide Time: 17:39)



Program Object

Other OpenCL APIs related to Program Object-

- **clBuildProgram:** Build (compile and link) a program executable for all the devices or a specific device(s) in the OpenCL context associated with the program
- **clCompileProgram:** Compile a program's source
- **clLinkProgram:** Link a set of compiled program objects and libraries and create a library or executable
- **clGetProgramInfo:** Return information about a program object
- **clGetProgramBuildInfo:** Return build information for each device in the program object

OpenCL Runtime System

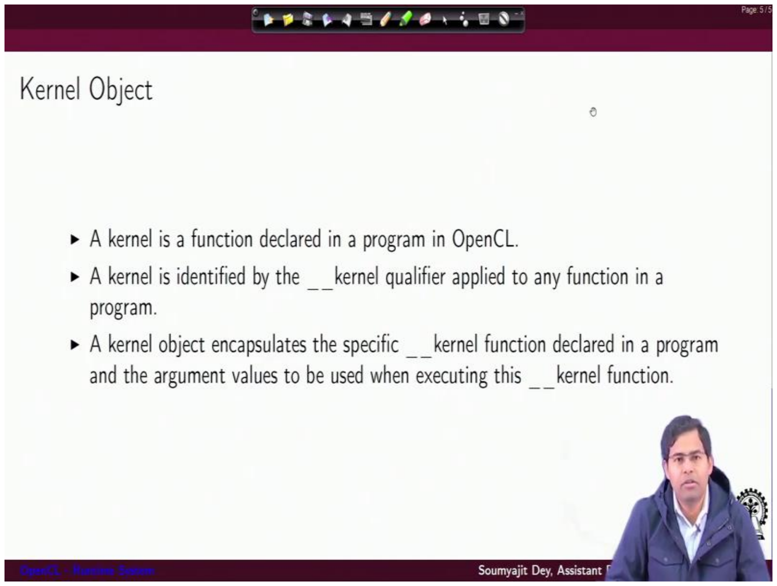
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, there are some other OpenCL APIs which are related to program objects. For example, you have CL build program so, what does it do? It essentially compiles and links a program executable for all devices or a specific device in the OpenCL context associated with the program and then FCL compile program which is compiling the Source Link program. So, essentially as you can see, you can build the overall process or you can program compile, and then you can link a set of compiled program objects and libraries.

And create the overall library or executable, then you have CL get programming info for which can actually return you back some information about an already compiled and created program object or you can see I will get program building for which will give you build information about the process that was followed and all that for creating the program object for each device. So these are like last.

So the first 3 will be coming I mean, standard compilation link programs, again used to be useful if you want to do these things by the host program. If you want the information extracted from the project that is available in the context in the runtime, then you would need to execute this last 2 commands.

(Refer Slide Time: 19:02)



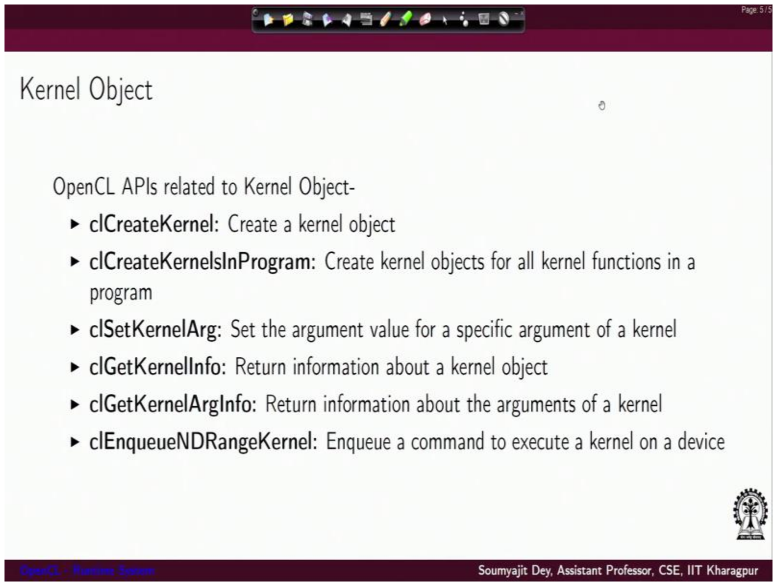
Kernel Object

- A kernel is a function declared in a program in OpenCL.
- A kernel is identified by the __kernel qualifier applied to any function in a program.
- A kernel object encapsulates the specific __kernel function declared in a program and the argument values to be used when executing this __kernel function.

Soumyajit Dey, Assistant

So what is the kernel object a kernel function is declared in the program? The kernel is identified by this kernel underscore kernel co qualifier as we have seen earlier. Now this corresponding kernel object encapsulates this function, which is declared in the program. And also it encapsulates the albumen fellows to be used when the kernel function would be executed.

(Refer Slide Time: 19:30)



Kernel Object

OpenCL APIs related to Kernel Object-

- **clCreateKernel:** Create a kernel object
- **clCreateKernelsInProgram:** Create kernel objects for all kernel functions in a program
- **clSetKernelArg:** Set the argument value for a specific argument of a kernel
- **clGetKernelInfo:** Return information about a kernel object
- **clGetKernelArgInfo:** Return information about the arguments of a kernel
- **clEnqueueNDRangeKernel:** Enqueue a command to execute a kernel on a device

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

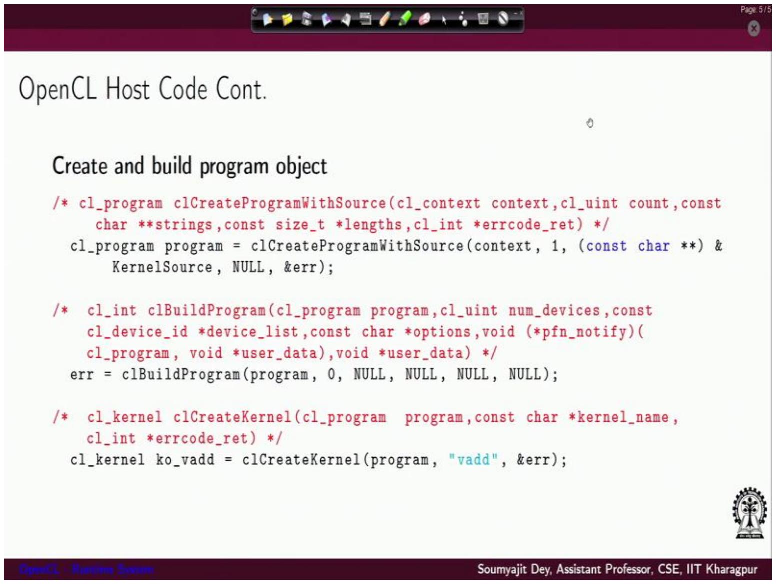
So just for manipulation of kernel type objects, you again would have be having some specific OpenCL API support. For example, you want to create the kernel object or you want to create kernel objects for all kernel functions in a program of course, you can have multiple kernel functions. And the next important thing is, if you want to set the argument values of a kernel. Now, what I mean you, we know that for a function we can wrap to specify a set of arguments.

Now the way it is done for an OpenCL Kernel is you fire multiple instances of CL said kernel argument to you with each instance you add in a new argument, for a specific argument of the kernel. And if you want to get information about the kernel object that will do that you can get from CL get kernel info, if you want argument information about the kernel you can get it from CL get Kernel Arg Info for return information about the arguments of a kernel.

And then if you want to enqueue a command, which will actually execute a kernel in our diversity in a context for that the actual command is CL enqueue indeed kernel so you are enqueuing substances and enqueuing of a command as we have understood, unlike CUDA well just launch a kernel here things are a bit more system specific. So even a kernel execution is a generic open a command.

So it is equally known by the name CL enqueue ND range kernel. Since it is a queuing operation, you do not really get to execute it directly you enqueue, an execution type command in a command queue. That is why you call it CL enqueue ND range kernel.

(Refer Slide Time: 21:19)



OpenCL Host Code Cont.

Create and build program object

```
/* cl_program clCreateProgramWithSource(cl_context context,cl_uint count,const
     char **strings,const size_t *lengths,cl_int *errcode_ret) */
  cl_program program = clCreateProgramWithSource(context, 1, (const char **) &
       KernelSource, NULL, &err);

/*  cl_int clBuildProgram(cl_program program,cl_uint num_devices,const
    cl_device_id *device_list,const char *options,void (*pfn_notify)(
    cl_program, void *user_data),void *user_data) */
  err = clBuildProgram(program, 0, NULL, NULL, NULL, NULL);

/*  cl_kernel clCreateKernel(cl_program  program,const char *kernel_name,
    cl_int *errcode_ret) */
  cl_kernel ko_vadd = clCreateKernel(program, "vadd", &err);
```

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

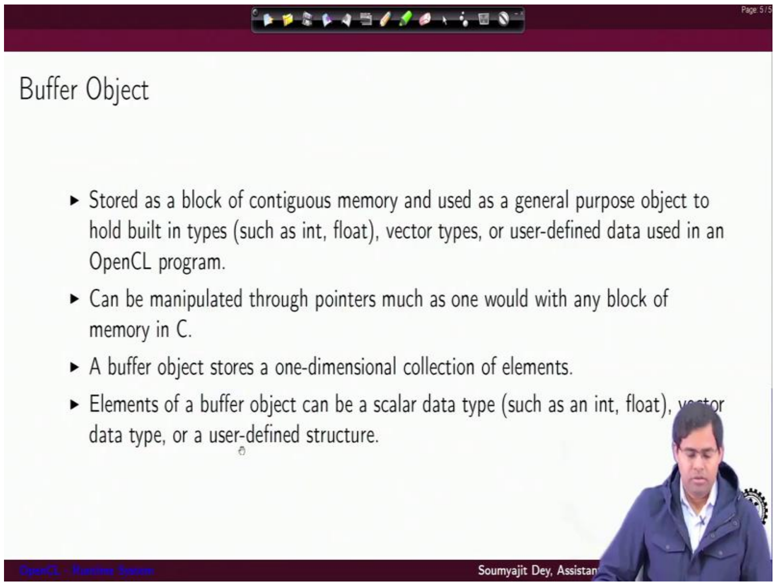
So let us see how the OpenCL host code actually continues from the point where we left where we just identified the platform ids and the divided device ids. So and also we created a context with a single GPU drive device. So the first thing we will be doing is we are going to build the current program object. So for that will be filing the CL create program with source command, where we will be providing this as you can see, ignored the other command but observe these are command which is the characters command.

So, essentially kernel source is like a character array, which is containing the actual source code of the kernel. So with this program you get we started with this function create program with source you are expecting that character to past content, the real source code and it is creating the program out of it. Now, once you have the program object, since in this is the type of competition we are following here that you are going to build a program object.

From the source that is why we fired this function as we discussed earlier, if you are already having the binary available, they are the intermediate code available the other options of CL create program would be used as we have seen earlier in this case, we are just showing this example and then once the program object is ready, you like to build it that is compile and link. So since the object is with name is program use of type CL underscore program these are type for program objects.

The next you execute this command to build it so, once that build of the program object is done, you have to create the actual kernel objects which are going to be enqueued for execution. So now you pass this program object that has been built to the CL create kernel command and it will return you this OpenCL Kernel type object ko vadd.

(Refer Slide Time: 23:36)



Buffer Object

- 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.
- A buffer object stores a one-dimensional collection of elements.
- Elements of a buffer object can be a scalar data type (such as an int, float), vector data type, or a user-defined structure.

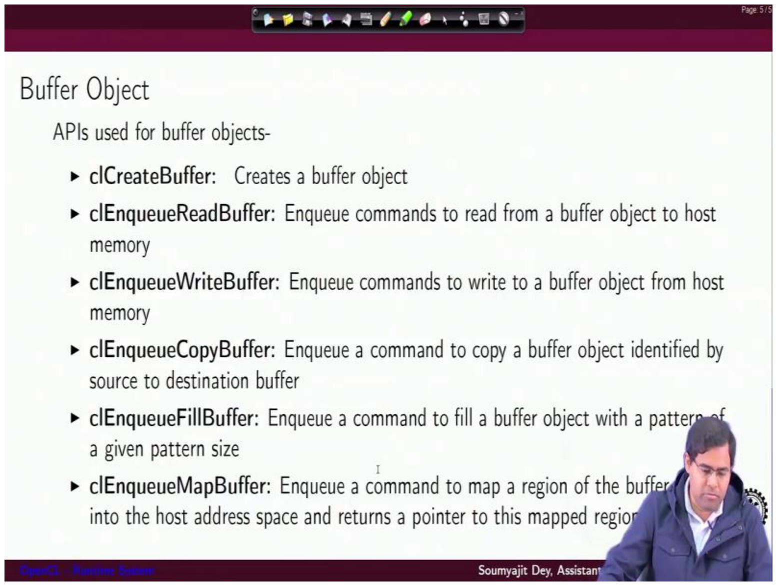
Soumyajit Dey, Assistan

So this is how you create a kernel object finally, using the CL create kernel, before that you build a program before building the program, you create the program object. So again, I will just repeat you we are considering that kernel is available as a source in a character array, use that to create program object. Use that to build the program use that to create the kernel once the kernel object is created, now, the other thing is just like kernel, you also need the memory space to be declared as buffer objects.

So, these are stored as block of contiguous memory like normal C. And they are used as general purpose object to hold built in types like in float vector types or you can also have a user defined type in open scale problem, they can be manipulated to pointers, much as 1 node with any block of memory in C and of course, also, it stores essentially it is a 1 dimensional collection of elements like CL is like elements of our object can be as scalar data type, or a user defined structure.

As we know now one thing I would like to point that most of this information we have mined out from the khronos specification, that is that is available in the khronos website. So that is quite openly available. Most of this information has been mined out from those sources and also the references we are provided at end of this presentations.

(Refer Slide Time: 25:01)



Buffer Object

APIs used for buffer objects-

- **clCreateBuffer:** Creates a buffer object
- **clEnqueueReadBuffer:** Enqueue commands to read from a buffer object to host memory
- **clEnqueueWriteBuffer:** Enqueue commands to write to a buffer object from host memory
- **clEnqueueCopyBuffer:** Enqueue a command to copy a buffer object identified by source to destination buffer
- **clEnqueueFillBuffer:** Enqueue a command to fill a buffer object with a pattern of a given pattern size
- **clEnqueueMapBuffer:** Enqueue a command to map a region of the buffer into the host address space and returns a pointer to this mapped region

Soumyajit Dey, Assistant

Now, if I am trying to create some buffer objects, just like for kernel objects, we have a set of commands to execute for buffer objects. Again, we have some commands to execute. So first you have the CL create buffer, which will create the buffer objects you have been buffer, to which you can enqueue commands pertaining to read or write to be done on buffer objects. So, for example, CL in queue read buffer, these enqueue commands to read from a buffer object to the host memory.

So that is important, you have the host you have 1 of the devices identified as your host device. So you want to read back data that has been computed on the device by an OpenCL kernel. Now the buffer object is pertaining to the memory region in that device in which the OpenCL is executed. So to read from that, to the host side memory, you have to execute command I will say that unlike a CUDA command for read/write to and from the host program, things are like here you have to enqueue or read command.

So you have to read from the device side buffer object to the host side memory you enqueue or read command CL enqueuer or read buffer, you have to write from the first memory to the device I buffer object, you will have a ceiling to write buffer to write to the buffer object from the host memory. Now, of course, I mean, you have to remember that we are not talking about normal GPU things here like CPU, GPU, best good execution, like it can also be that the host and the device have a shared memory space on which things are going on.

We will see some examples of such kinds of devices later on now coming here, so apart from CLN, to read and write buffers, you also have this CL enqueue copy buffer. So essentially, you can copy a buffer object identifiable source to destination or some destination buffer, you can also feel a buffer with a pattern if you give a pattern and the pattern size. And also you have the map operation to map a region of the buffer object into the host address space and return a pointer for this map region.

So, you have a region of the buffer object which you can map to the host address space and return the pointers. So that is also something that will be possible. So these are the different buffer manipulation commands that are there maybe we this will like to end this presentation. Thank you.