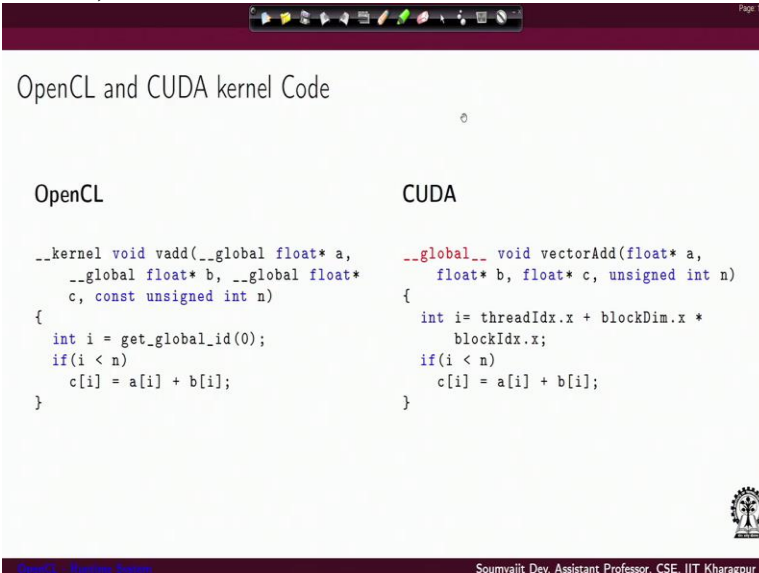


GPU Architectures and Programming
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology-Kharagpur

Lecture # 43
OpenCL - Runtime System (Contd.)

Hi, welcome back to the lecture series on GPU architectures and programming. And so, in the last lecture we have just introduced the example on OpenCL and CUDA kernel if you remember.


(Refer Slide Time: 00:36)



The slide displays two code snippets side-by-side. The left snippet is labeled 'OpenCL' and the right is labeled 'CUDA'. Both snippets implement a vector addition function. The OpenCL version uses `__kernel` and `__global` qualifiers, while the CUDA version uses `__global__`. The OpenCL version uses `get_global_id(0)` to determine the thread index, whereas the CUDA version uses `threadIdx.x + blockDim.x * blockIdx.x`. Both versions iterate over the first `n` elements of arrays `a` and `b` to calculate the sum and store it in `c[i]`.

```
OpenCL and CUDA kernel Code
```

OpenCL	CUDA
<pre>__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]; }</pre>	<pre>__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]; }</pre>



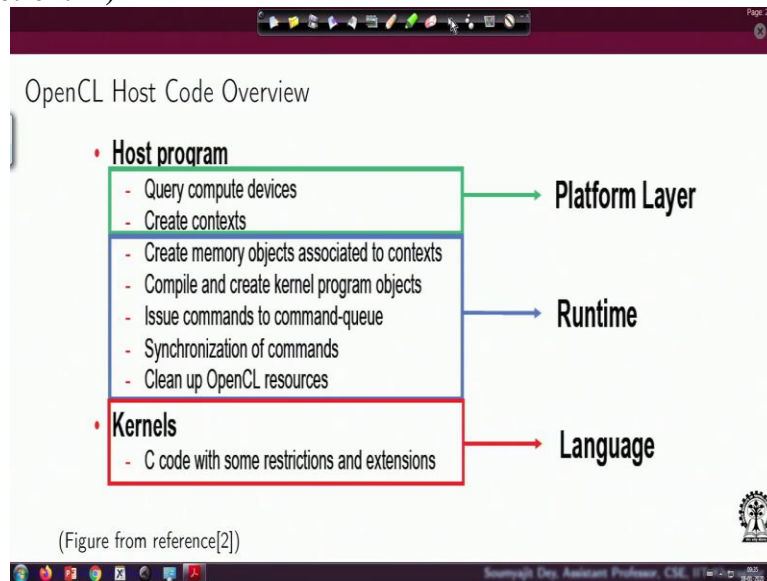
OpenCL - Runtime System Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

And we have just shown a simple Vector as an example, where we are side by side showing examples of OpenCL and CUDA kernel just to identify what would be the differences in the way things are to be written when you are creating an OpenCL kernel. So, from this 1 thing is for sure we understand that both they are kind of extensions of C and when we come to OpenCL, there are several variants at that at the lowest level you have the OpenCL specification which is being implemented in the form of a C library that can execute in a parallel manner in the suitable devices.

And at the high level, you also have C++ based wrappers and at the end serve a phi OpenCL, which is a higher level wrapper, which makes the job of writing OpenCL kernel is far easier at least writing the host programming specifications for easy for our purpose since we are living with a C background in this course, we are assuming on with that so, we are looking into

everything at the lowest level that how OpenCL C library and a specified what are the function calls which are used for different functionalities.

(Refer Slide Time: 01:47)



So coming back, if you just recall the way we discussed the different layers through which OpenCL code executes. So every program every OpenCL program comprises this host program that will start with some platform layer level queries. Because in the runtime system, you have the platform layer followed by the runtime. So, these are essentially what is going to the platform layer to let know, what are the compute devices available in a system?

And again, I will say that this is the way we are going to write a genetic code. Of course, if you are going to write a simple, OpenCL code, which is specific to some device, your code can be much simpler. So initially, we will be querying the compute devices and based on this query will get the result how many different differ, I mean, how many compute devices are really there and what are the different vendors from the come from each other, I mean, what is organization of the compute device which have them sitting the same and all that.

And based on this information of computer devices, you can decide that you will be kind of accumulating a set of devices together to define what we call as a context essentially, that a context is an abstraction through which multiple devices can share their data. If you have segregated a set of 2 sets of devices into 2 different context, they would need to communicate

among each other through the host program without direct without having any direct scope of communication or synchronization among them.

Now, once you have queried the compute devices, understood their organization and created the context in the platform layer, you move on to the runtime where you actually so this is all happening in the simple in the 1 program, you have to do all these things followed by the followed by execution of the corner. So we are just trying to specify the structure of the program here. So these are the initial structures and now they will be followed by the way different memory objects are to be created and they are to be associated to the context.

So, this is the part of code which should come immediately after you create the context. Once this is done, and you I mean, and we also assume that you have the kernel spin speed written, and the typical way in which the host program will be managing the kernel, where this is something really a bit different. So, essentially, we will be capturing the kernel like a string, let us say in a character edit. So, once that is done, there will be OpenCL kernels, which will be used to come to create object.

A program object from that character string, which is containing the kernel and with this, you have the entire setup ready. So just to summarize, again, you have query devices, you have segregated devices belonging to different vendors in architecture specific way, so that you can create. So that there is the idea of platforms to maybe having different platforms content together, using the platform in the platform layer.

After you query the devices, you create context. Of course, 1 context again, this is also important that in 1 context, typically, it would be using dead vendors OpenCL implementation, you may have 2 different contexts sitting in 2 different vendor provided devices at 2 different vendor provided platforms containing multiple devices. So this is very important, you have to understand, you can have multiple platforms coming from different vendors.

And each platform can contain multiple compute devices of CPU, GPU, maybe some other compute nature, like tsp and all that. So, that is why all this complexity at the lower level

program can come in. But you have to also understand that this gives you a lot of power with trust with respect to deciding how to really program such a heterogeneous device. And as we move on the importance of heterogeneous devices, will be clear, very soon.

So you query the devices, you understand the entire setup of the system that is you understand, which are the different vendor supplied devices, or platforms inside the platforms? How are the devices organized? Based on that you can have, let us say you have 4 devices in a platform you can create, I am just giving an example subsets of 2 devices to create 2 different context. Or you can bunching all the 4 devices together to create a single context and like that.

And then in the next part here, what we have is the things that you do after the contexts are created across different platforms, or if I am having a very simple architecture, I may be creating context inside a stem platform. And then you create data objects associated to the different contexts. And then you create kernel object based on a kernel specification, which is encoded inside a character array and that is compiled to create the kernel objects after all this what you have is you have programs steady as objects.

Which will do the compute and you have data segments also setup as memory objects. So, that are containing the data on which the programs are going to the executables are going to compute. So, what is the logical next step the next step would be that you have to set up a sequence of commands which are to be executed what do I mean by commands comments means when and how to write, run which kernel on what memory object produce the result, it to which memory object followed by which kernels execution, all this stuff.

So, to give you an example, let us say I want to implement the following computation. I have kernel A and kernel B they are produced data. And that is going to do buffers. Once the data is ready, then I want some kernel C to fire and produce the data and moving into a buffer. And you have these input buffers. So I have a sequence of operations. So I need to write the data into these buffers. The most important you need to remember that initially, you do not have you just have these buffers there.

And you have these kernels available to you as program objects. These are available as memory objects, buffer objects. So you need command to write data here. You need command to execute these kernels you need command who will let you know that these kernels have executed and produce data. And then you need another command to execute this kernel. And again, you have to quit the system to know that this kind of execution is done so the data is ready here.

So that means I need a sequence of commands somewhere, because if the commands are set up in a wrong way, then everything is going to go wrong? So, this leads me to the idea of 40 call as a command queue, through which all these comments will be sequenced. And that will provide me and specific order in which the operations have to be carried out. So that this required functionality is actually implemented. These are important.

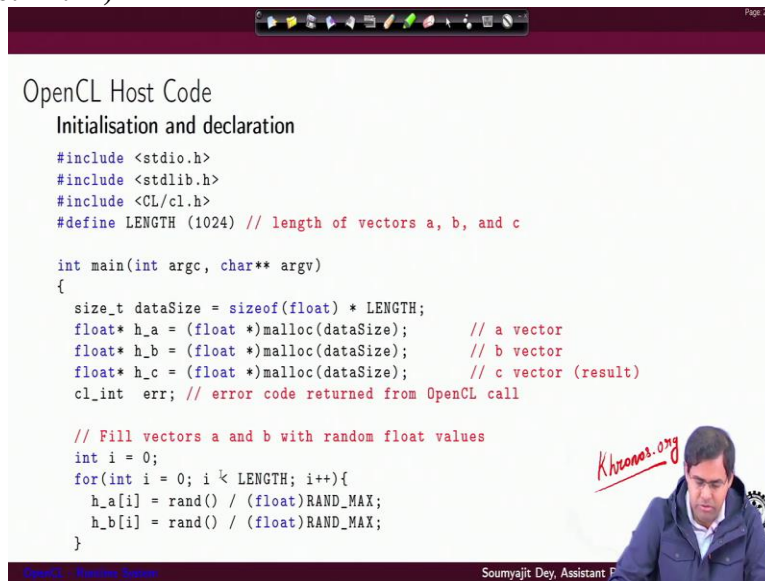
So I am just trying to motivate why I need this comment queue, this idea of a comment queue. Because unless I have a queue like structure for storing comments, I do not have any way to force this kind of dependencies and sequences in the execution,? Of course, I also need something else. Which is for example if I take this thing that this guy is going to execute only when the data is ready here and here.

So this would be naturally like a synchronization point in the execution. So how do I specify which are the points where the programs need to synchronize? That would require not only execution comments, or read or write comments, but also synchronize comments. So in this way, I am just trying to motivate what are the different kinds of comments that are going to be part of the comment queue.

So once this is done, let us say you get the data ready here, you know that things are done, you need to clean up the resources that you have consumed in a system that would mean you need to clean up this well, first, you need to clean up this program. I have just done everything. So in what form is essentially a colonel present as we have said that finally inside the host program, the colonel has to be available like a string of extra character type array.

In general, when we are specifying the kernel is just like a normal C code with some restrictions and extensions. There by then I mean typically there will be some OpenCL constructs which has been pushed in the form of the functions which are implemented by the runtime. And also you have to understand that these are data parallel code. So essentially in the C code what you write is all portrayed via just like a CUDA function.

(Refer Slide Time: 11:24)



```
OpenCL Host Code
Initialisation and declaration

#include <stdio.h>
#include <stdlib.h>
#include <CL/cl.h>
#define LENGTH (1024) // length of vectors a, b, and c

int main(int argc, char** argv)
{
    size_t dataSize = sizeof(float) * LENGTH;
    float* h_a = (float *)malloc(dataSize); // a vector
    float* h_b = (float *)malloc(dataSize); // b vector
    float* h_c = (float *)malloc(dataSize); // c vector (result)
    cl_int err; // error code returned from OpenCL call

    // Fill vectors a and b with random float values
    int i = 0;
    for(int i = 0; i < LENGTH; i++){
        h_a[i] = rand() / (float)RAND_MAX;
        h_b[i] = rand() / (float)RAND_MAX;
    }
}
```

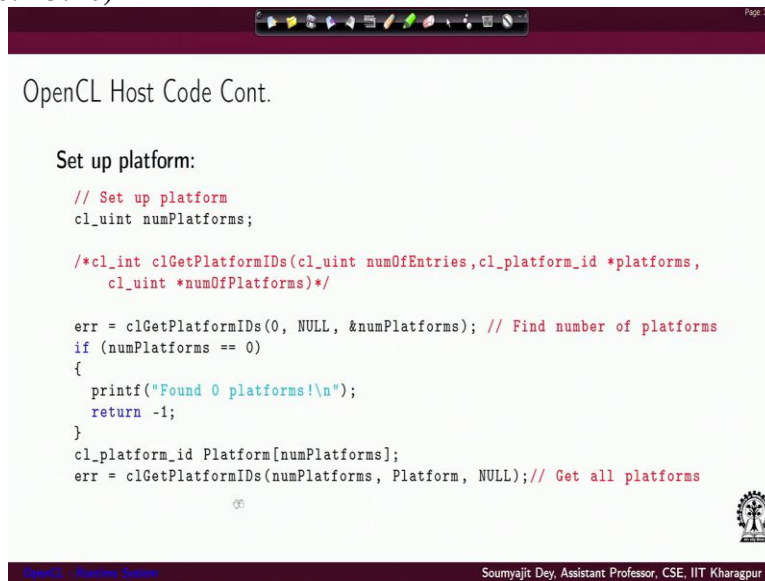
So, with this will first take an example of our host code. So, this was our example of a kernel code. And now, we will take an example of a host code. So, essentially is just like a normal C program as you can see you include `stdio.h` type `std` under library dot `h` and then you have OpenCL specific declarations in `cl.h`. So, these are the declarations that are kind of specified by the working group. The joint working group called khronos.

So you may visit this website. So this is essentially the Joint Working Groups website, who actually specifies all the standards and specifies all these functions for OpenCL. And what will be their behavior and all that. Now, whenever some vendor will conform to this behavior, they will provide the library which will contain the implementation of the functions which realize this behavior.

So, let us just have a look into the program like what is really going on. So as you can see, the initial part is just like a normal C program you are declaring 3, you are just articulating 3 spaces using memory spaces, you are `malloc` in them, allocating their memory for `h_a`, `h_b` and `h_c`

corresponding to this data size? The length of the vector that you did like to work on. And then let us say we are just using a code to fill up these vectors by randomized initialize values.

(Refer Slide Time: 13:10)



```
OpenCL Host Code Cont.

Set up platform:

// Set up platform
cl_uint numPlatforms;

/*cl_int clGetPlatformIDs(cl_uint numEntries, cl_platform_id *platforms,
    cl_uint *numOfPlatforms)*/

err = clGetPlatformIDs(0, NULL, &numPlatforms); // Find number of platforms
if (numPlatforms == 0)
{
    printf("Found 0 platforms!\n");
    return -1;
}
cl_platform_id Platform[numPlatforms];
err = clGetPlatformIDs(numPlatforms, Platform, NULL); // Get all platforms
```

OpenCL: Building Systems Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Once that is done, now comes the platform specific programs. So let us have a look at how this platform is created is very important. Please pay attention here we from this is the point from where things become a bit different from our normal C program running in a desktop computer. So, first, you see the usage of OpenCL type here. So you have a CL underscore uint for OpenCL kind of unsigned integers.

So that would be storing the number of platforms. So you are initializing a variable num platforms for storing the number of platforms that would be available in a system. So forgetting that you make a call to this function, clGetplatformIDs. So, this is the function which are calling. Now this is that definitely that is the kind of declaration we have put in the commands there for the function.

So, as you can see in this commands, we are specifying that this is asking for the following the number of entries and then you are hoping that it will actually give back the entries. And so that would be coming in another output, which is the cl uint type point it is a pointer to number of platforms. And then also there is another argument we will see what why this is useful. So, first thing you do is you make a call to clGetplatformIDs with the number of entries the first parameter set to 0. And the second parameter which is the type platform ID.

So that is another OpenCL type for platforms. There is also set to now and you are just passing the address of this variable, which is a cl you unsigned integer type, which is the non platforms variable. All that is going to happen is, if this call is set up in this way, with the first parameters being 0 and now, it will essentially write the total number of platforms into this variable. All that is all literally, suppose there are only 2 platforms available, it does very well we will get a valid 2, if it is only 1 platform, which will get a value 1 like that.

In case is 0, we have written some handler code that then it will just return back from here because it is not able to find me is finding 0 platforms there are no platforms at all. Now, suppose you have figured out through this call that what is the number of platforms here? Now you are going to use exactly the same call again, to figure out what are the different platforms I am because just knowing what are the platforms are not going to I mean how many platforms you have is not going to help you.

You need more information like how to access each of these platforms for that, you again make a call to the same function, you can see the functions of sim clGetplatformIDs, but the arguments are different. Initially you give the first argument is 0 because you simply did not know. So, that is like an initial value you are giving. But now, you are you know, how many platforms are there through this style of call to the function. Now you are calling it in a different style, you are supplying it the value of number of platforms.

And you are, setting up something here which is some you are passing the variable which would be platform and its type is cl platform id. So of course that needs to be declared here, and that is something you have to do. So, suppose you have this. So, you have an earlier it was said to now, suppose you have already declared this, that what is here platform id, and then you have given this variable name, this is essentially a pointer out here.

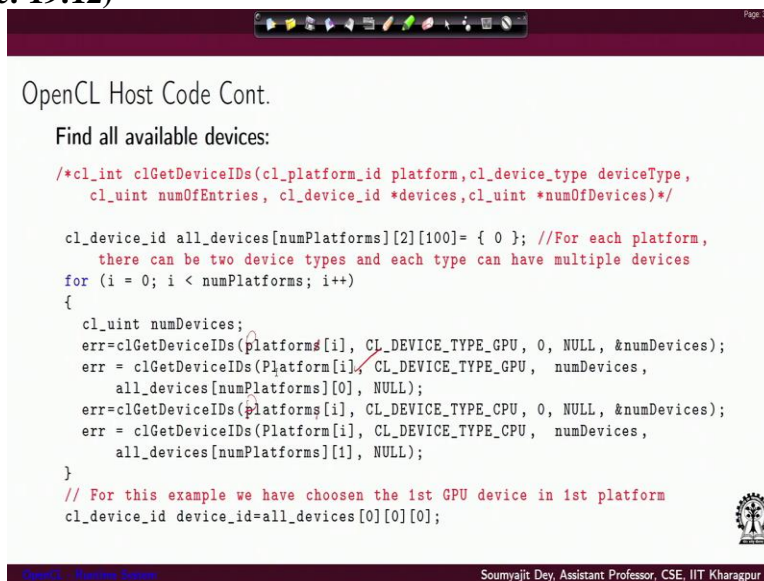
And you are now setting the last thing to now, because now you want this function to act in a different way, because you already know how many platforms are here. But you really want to copy back these platforms ids to some structure, which is of type cl platform id. We have been

declaring each here. So you have cl platform id type for this variable platform, which is essentially add a and b by knowing a proper initialize value of nun platforms, you are able to not declare.

So, this is like a dynamic era you are declaring, you are able to declare what should be the size of the platform at it. And you are so in a way, I mean, using that size, which is now known to you, you are simply declaring platform of size number of platforms. And it is of type platform id, because you expect this array to contain the number, the num platforms, number of platform ids. So, you have these many entries, each of the entries should be a valid platform id.

So when this call will be set up in this way, and it will if it returns without any error, then this platform array will get filled up with all the device the all the ids of the platforms in each of these positions. So you can see that you are using clGetplatformIDs to first recover the number of platforms that are there. Accordingly are declaring the dynamic edit. And then you are populating that area in different each of the positions in the area with the platform IDs, which are being recovered using the second call.

(Refer Slide Time: 19:12)



```
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(platforms[i], CL_DEVICE_TYPE_GPU, numDevices,
    all_devices[numPlatforms][0], NULL);
  err=clGetDeviceIDs(platforms[i], CL_DEVICE_TYPE_CPU, 0, NULL, &numDevices);
  err = clGetDeviceIDs(platforms[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];
```

OpenCL, Runtime Issues

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, let us inspect this code very minutely, this has been written in this way just to give you a general feel, how like how things occurred there. So, you know, now, what are the platforms using their IDs? But as we know that in OpenCL, we are assuming that the underlying system is very complex, like there are the things from multiple different vendors. And each of these

platforms may contain multiple devices, like 1 CPU, 1 GPU, or multiple GPUs, multiple CPUs in 1 device in 1 platform and again in another platform multiple such devices and all that.

Till now all that you have got is the platform ids. So now, you would like to get the device IDs for each of the platform IDs for this you are going to use this function, `clGetDeviceIDs`, we have provided here the prototype of this function. So, it will be taking the following parameters, you are going to ask the system, what is the device ID is available for some specific platform. So, the first parameter is of type `cl_platform_id`, where you are going to push the specific entry of this earlier platform at it.

And then you are also going to provide it with the type of devices that you wanted to discover. Maybe you were interested in only a set of only a specific type of device like CPU or GPU, then this will give you those device ids. And then you also have an entry for `cl` a number of entries. You provide it a pointer of type `device_id`, which you wanted to figure out and provided the device id there and then you have another pointer or number of devices.

This is also a type `unsigned int` let us see how things get set up. See now that we have identified that what is the number of platforms. So, using that. So, this is the number of platforms, which you have now figured out from this call, and you have declared this and you have figured and filled up this area platform with a different platform ids, you make the declaration of another 2d array called `all device id`.

So the use of types here `platform id`, these are type `cl device id`, why is that the 2d added because for each platform, you can have technically multiple devices. So we are just making a declaration like this. I mean, these are static is a reason. So for each platform, we are expecting in architecture, the devices are of 2 types? So we are writing it 2 of course, this will this is not generally I am just repeating.

We are simply assuming in this case that the devices are only have type CPU and GPU. We make it to, and we do not and we are giving an upper bound on the size here, which is also statically defined. So for each platform, there can be 2 device types. And each type can have

multiple devices, which would be stored in these positions? Now let us see how that thing works. We are going to browse through acquiring the system.

And we will for each of the platforms, we will query the system for device IDs and we will get the device IDs there is a code that will go into run and this is how it works. So we have a follow through which will run from $i = 0$ up to the number of from I mean, it will iterate for number of platforms types, each of the values of the iteration, what is going to do is you so suppose you are writing for 0.

So essentially you have got the first platform ID at platforms 0 position of the array. So, now you fire the `clGetDeviceIDs` call using this call you have you have also provided it with let us say platform 0. So, the first entry with the platform id and decided for this platform you are trying to discover devices of type GPU and here you also pass it with the num devices which has been declared here immediately before the call.

So in every iteration you will get a phrase variable here called num devices. So with num so, this will give you back the number of GPU devices that are available in that from 0 am speaking of things going on when the $i = 0$ the iteration. So with this call, so as you consider, the structure is again similar, you have the same call, same kind of call make twice. First you kind of discover what is the number of GPU devices that you have in this platform and then you make the second call again with num devices initialized. So it looks similar platform.

And then so just pardon us, we have this being inconsistent like this is platform and so this should be let me make the correction here. So platforms I, whereas the issue is, as you can see, I have we have here declared platform with P caps. So let us just follow that here. And I believe you should be able to resolve it. This is fine. So, with this first call, we get the number of GPU devices in platform in the first platform, then we provide that value here. And we are expecting that this system earlier, as you can see that 2 of these arguments were 0.

And now, you are actually setting them up the first argument, because if I am initializing the number of device to 0, now I know the number of devices, so I put it here. And then I want those

device IDs to be returned? So since my queries, what are the number of devices of type GPU in platform 0? Now I know that now I want to know that what should be and what are the device IDs for those GPU type devices in platform 0 so essentially here and providing it the point at 2 all devices, num platforms. In place of 2 labs 0, the let us say the first part is for storing the GPU IDs.

And then so that would be the pointer. And so I would expect with this call, this array would get populated with the different GPU ID device IDs for platform with ID 0. Now, in the next again, I will just repeat the same thing. So again, you are fighting the get device IDs call? Using these get device IDs called you are figuring out what are the what are the more different I mean, how many GPU CPU type devices are there in these platforms.

How many CPU type devices are there, in this platform, we were essentially figuring in platforms 0 and then for all those devices, you are storing back their device ids? Inside this all devices, non platforms, CPU for is just it will have one here. And all the rest of the positions in the array will be filled up with the CPU device IDs? As you can see, this is a 2d, the 3d array. So if I am associated 2d level essentially have a pointer.

We and we since have passed it, the function will ride back inside this array sequentially in these positions like num platforms 1 0, num platforms 1 1 in all these positions, it is going to ride back the CPU device IDs for platform 0. So with this iteration of this loop, I have discovered as it iterate through all the platforms which have discovered all for each platform, device IDs for CPUs, device IDs for GPUs, and all that together. For this example we have chosen the first device to be a GPU device in the first platform. So, maybe we this part will be ending this lecture. See you soon. Thank you.