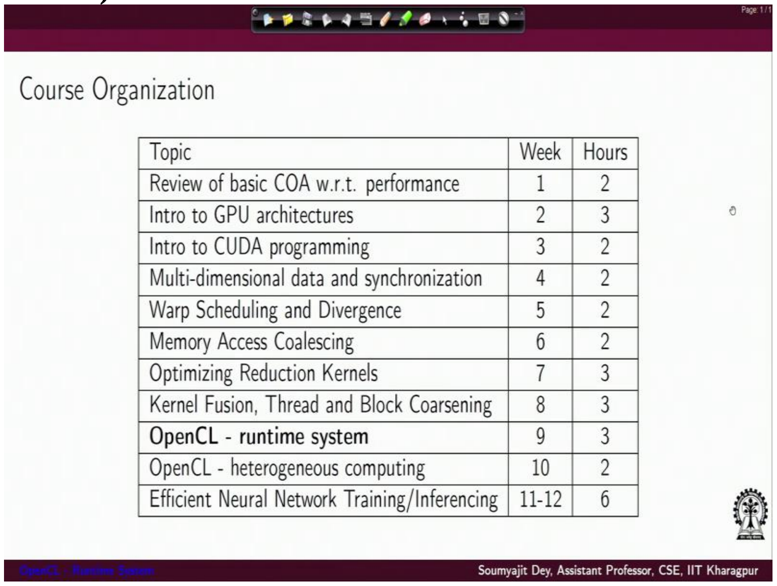
# GPU Architectures and Programming
**Prof. Soumyajit Dey**
**Department of Computer Science and Engineering**
**Indian Institute of Technology-Kharagpur**

**Lecture # 41**
**OpenCL-Runtime System**

Hi, welcome back to the lecture series on GPU architectures and programming. So from this lecture onwards.

**(Refer Slide Time: 00:31)**



## Course Organization

| Topic | Week | Hours |
|---|---|---|
| Review of basic COA w.r.t. performance | 1 | 2 |
| Intro to GPU architectures | 2 | 3 |
| Intro to CUDA programming | 3 | 2 |
| Multi-dimensional data and synchronization | 4 | 2 |
| Warp Scheduling and Divergence | 5 | 2 |
| Memory Access Coalescing | 6 | 2 |
| Optimizing Reduction Kernels | 7 | 3 |
| Kernel Fusion, Thread and Block Coarsening | 8 | 3 |
| **OpenCL - runtime system** | 9 | 3 |
| OpenCL - heterogeneous computing | 10 | 2 |
| Efficient Neural Network Training/Inferencing | 11-12 | 6 |

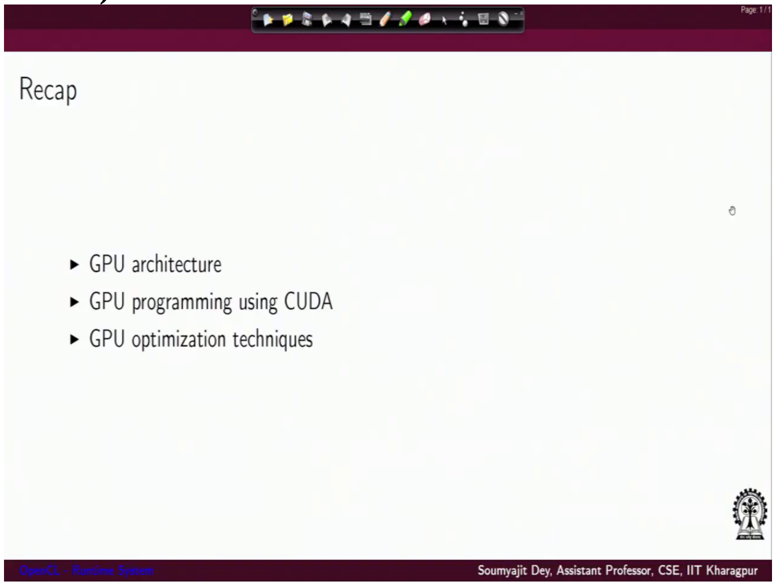
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

In week 9, we will be moving out of CUDA programming of GPUs. And we will be introducing an alternate programming paradigm which is also fairly useful in the context of GPU programming and this is what we call us OpenCL programming language. So, the way we will be making the coverage is that we will be introducing OpenCL as a programming language. But also, we need to understand that it has an underlying runtime.

So we will look into how the runtime system operates. And the next important thing would be will be looking into how OpenCL unlike CUDA, which is useful only in the context of Nvidia GPUs, we will like to see that how OpenCL became being a more kind of open, I mean, Fender independent offering how it is applicable for a wide variety of GPU or GPU like programming approach. I mean, computer architectures.

And that is why we will see that how OpenGL is specifically suitable for what we know call as heterogeneous computing. That means 1 can write an OpenCL program, which can be partitioned and schedule mapped and scheduled on a complex system comprising different computing blocks, starting from CPUs, GPUs, as well as SPGs.

**(Refer Slide Time: 01:50)**



Page 1 / 1

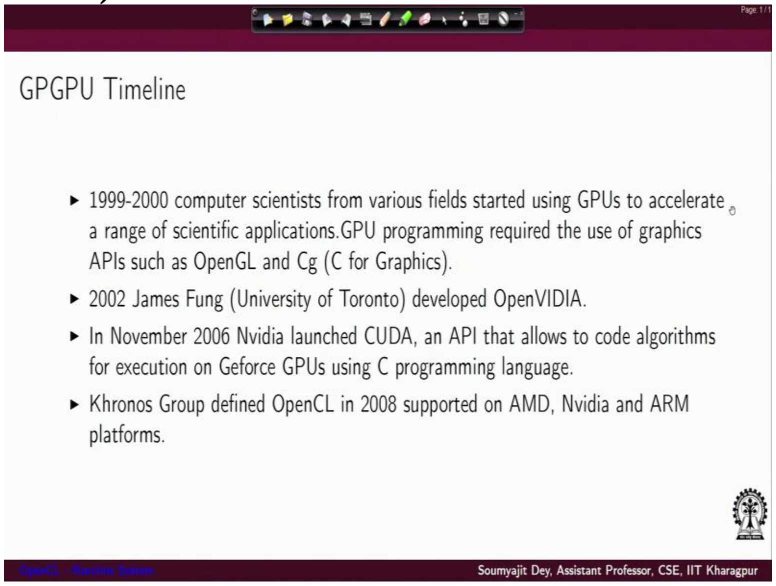
## Recap

- GPU architecture
- GPU programming using CUDA
- GPU optimization techniques

OpenCL - Runtime System

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

**(Refer Slide Time: 01:50)**



Page 1 / 1

## GPGPU Timeline

- 1999-2000 computer scientists from various fields started using GPUs to accelerate a range of scientific applications.GPU programming required the use of graphics APIs such as OpenGL and Cg (C for Graphics).
- 2002 James Fung (University of Toronto) developed OpenVIDIA.
- In November 2006 Nvidia launched CUDA, an API that allows to code algorithms for execution on Geforce GPUs using C programming language.
- Khronos Group defined OpenCL in 2008 supported on AMD, Nvidia and ARM platforms.

OpenCL - Runtime System

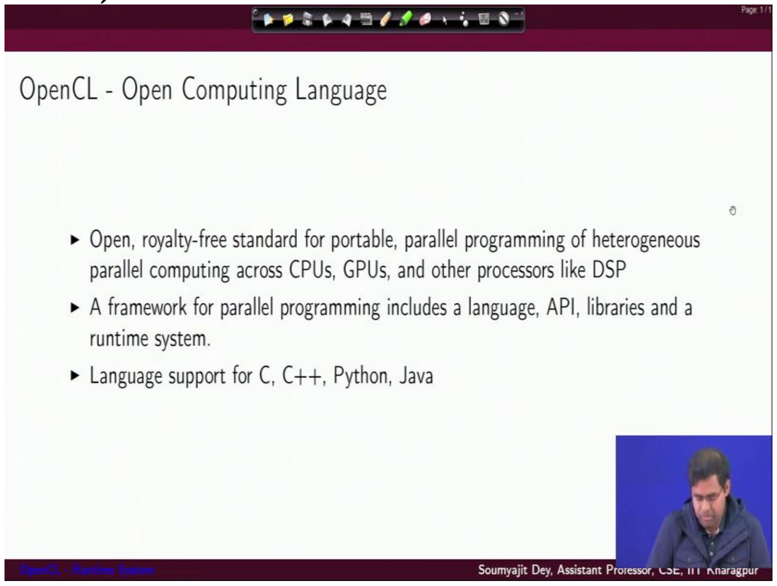
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So with this background if we just look back into how the idea of OpenCL came in, so in the 9 in this time around 1999 to 2000, computer scientists from various fields have already started using GPUs to accelerate a range of scientific applications. And, also, we know that if I am looking specifically into the case of graphics workloads, then also GPUs were particularly developed targeting those kinds of workloads.

And if you look into this idea of who was the primary vendor for GPUs that would be Nvidia around that time. And then people also start to think of alternate programming languages. Around 2002, James Fung from University of Toronto developed OpenVIDIA. And the way CUDA came into the picture was that in 2006, November, and Nvidia launched the CUDA API, which actually allowed due to write a program that would execute on our G force GPU.

And finally around 2008 this group, so essentially, it is a conglomeration of people started coming from different companies, the Khronos group, they came up with the specification of OpenCL. And it was supported by some of the leading semiconductor vendors like AMD, Nvidia, and ARM.

**(Refer Slide Time: 03:28)**



OpenCL - Open Computing Language

- Open, royalty-free standard for portable, parallel programming of heterogeneous parallel computing across CPUs, GPUs, and other processors like DSP
- A framework for parallel programming includes a language, API, libraries and a runtime system.
- Language support for C, C++, Python, Java

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So what opens here, first of all, the full form would be open compute language or open computing language. So essentially, it is an open royalty free standard for portable parallel programming of heterogeneous parallel computing or I mean portable parallel programming for a wide range of systems like CPUs, GPUs, and also other kinds of processors are, for example, DSP processors. Now the idea is that we have seen how CUDA works.

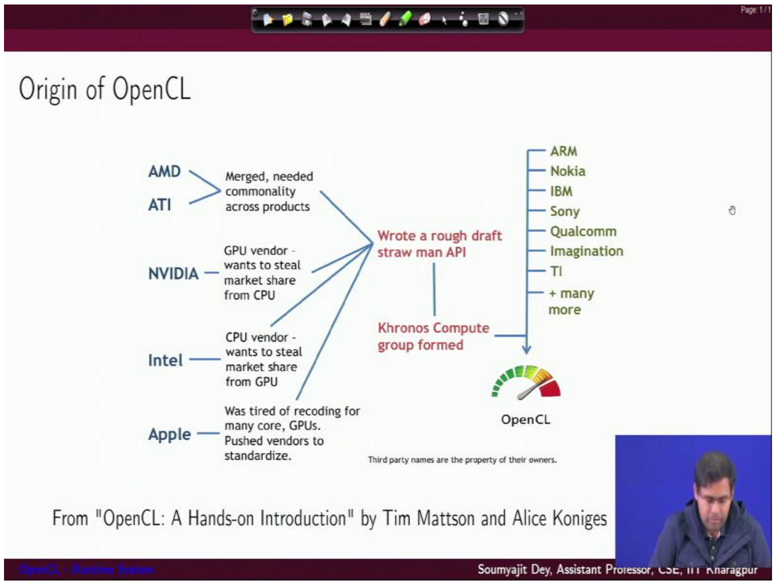
So essentially you write data parallel kernels and you execute them in GPUs, the but just like GPUs have got a way to execute assembly code. And we know that CPUs also have that facility, although at a limited level of parallelism. Similarly, there are also other processors like DSPs, who also provide facilities like parallel data parallel programming execution, of course at a level that is not much not that much paralyzed like GPUs.

But the idea was there since this kind of data parallel compute is supported by other architectures also, why not we come up with a specification such that any semiconductor vendor if they develop drivers, which satisfy their specification, then the program should run as is on those platforms. So, OpenCL in totality it comprises the definition of OpenCL Language, its application programming interface, the API, the associated libraries, and a runtime system.

The runtime system will manage the dispatching of the code. I mean, and interfacing with the lower level drivers and all that. Also, OpenCL provides language support for C, C++, Python and Java. That means that would mean that essentially the OpenCL borrows I mean, it is essentially the underlying thing is all defined on C, but for the providing somebody it will ability to give some high level specification like from a language, which is much easier to understand or easier to in terms of specification.

And there are Open CLs C++ as well as Python binding like phi OpenCL, which are also useful for writing OpenCL code. So that one need not write the code and add the much at the level of detail that is required if you are writing an OpenCL program in the original semantics.

**(Refer Slide Time: 06:10)**



## Origin of OpenCL

AMD
ATI
Merged, needed commonality across products
NVIDIA
GPU vendor - wants to steal market share from CPU
Intel
CPU vendor - wants to steal market share from GPU
Apple
Was tired of recoding for many core, GPUs. Pushed vendors to standardize.
Wrote a rough draft straw man API
Khronos Compute group formed
ARM
Nokia
IBM
Sony
Qualcomm
Imagination
TI
+ many more
OpenCL
Third party names are the property of their owners.

From "OpenCL: A Hands-on Introduction" by Tim Mattson and Alice Koniges

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, how did really OpenCL come up with first of all these comments and things that are written here and not my personal opinions, but they are more like so, I put in the difference from where I have taken up this idea that how the evolution of OpenCL happened. So, for example, there were a series of events which took place for example, the company that is AMD got merged with ATI. Now, ATI some famous for creating local strophic skirts and AMD brought ATI and created his own line of GPU architectures on the in parallel.

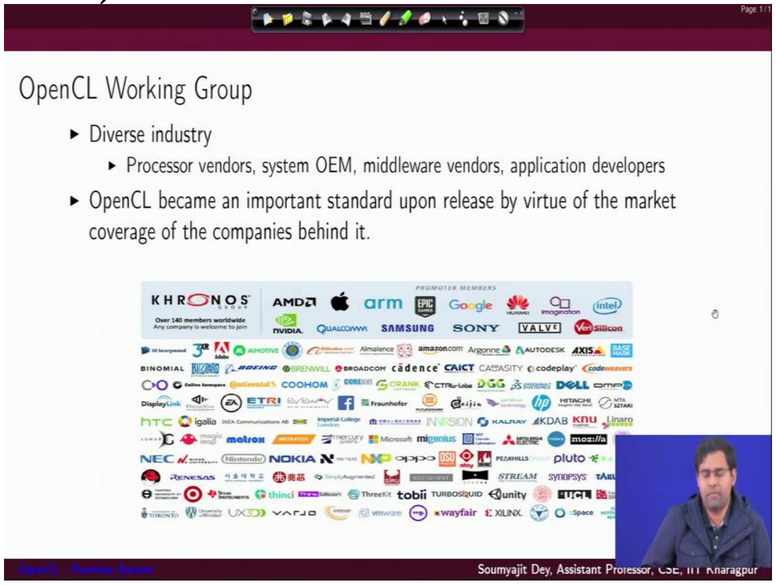
Nvidia was also working and it has come up with already I mean, Nvidia GPUs which are already in the market and have drawn a significant amount of market share from CPUs because people soon found that these GPUs were useful for running some parallel compute, unlike I mean, they were giving it giving much higher throughput with respect to a comparable CPU. But that would also mean that Intel CPU vendor like Intel will like to get people back into their fold by prover by supporting a language.

So that if you write GPU oriented kernel in that language, it will also run seamlessly in the CPU maybe with a different performance factor. And some other company like apple at that point of time was also advocating usage of OpenCL. So all Apple products came with OpenCL library already built in so that if you are writing an OpenCL code that would be that can just run like off the shelf on any Apple system.

So with support from all these meters may some vendors the khronos group was formed who kind of develops the specification of the OpenCL language and they formulated the basic draft of OpenCL. And also it received support from a significant number of other companies like embedded suppliers like ARM Nokia and IBM Sony Qualcomm was also 1 of the supporters and TI and there are many others who supports OpenCL.

So, when I say there are companies who support open CL that would mean that whenever they come up with a silicon, their own architecture process the architecture associated with that they will also provide a library, a low level library and the required drivers called the ICD files using which, if somebody is writing an OpenCL program, written as part I mean the return satisfying the original khronos specification of OpenCL programs, then that library would support the execution of such a program on the respective platforms.

**(Refer Slide Time: 09:09)**



OpenCL Working Group

- Diverse industry
  - Processor vendors, system OEM, middleware vendors, application developers
- OpenCL became an important standard upon release by virtue of the market coverage of the companies behind it.

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, this is also a figure which has been borrowed from the existing literature, which kind of shows that OpenCL is supported by a diverse set of silicon vendors system OEMs, middleware vendors, application developers, and it is and due to this huge popularity, I mean, it is of course, I mean, important standard, just because of the set of companies that are actually in a support. Now, of course, one natural question would be that, I believe there will be 2 important questions like, we always hear about Nvidia, GPUs and GPUs.

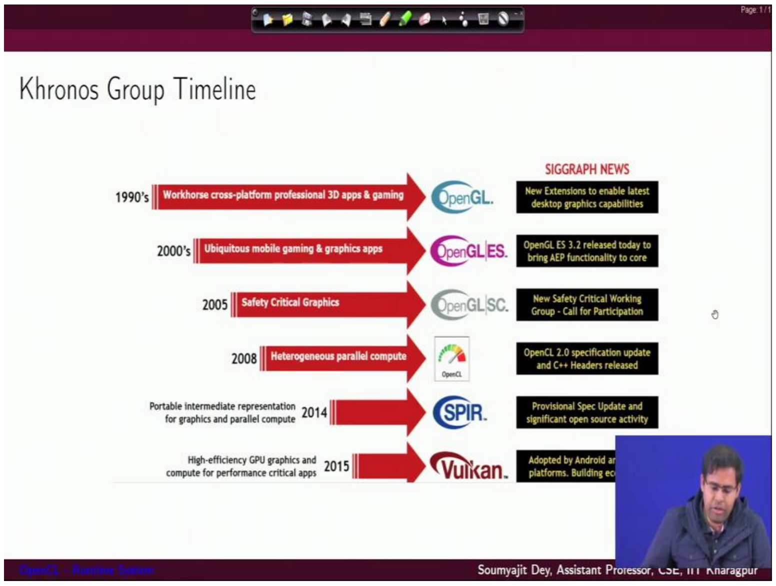
But we are not that much familiar with OpenCL as a programming language. First of all, the reason is, I would say, sensitive stuff fundamentally C base programming language and also the specification is a bit more complex, maybe I mean, in my personal opinion writing and OpenCL program using the lowest level CPI would require much more a bit more effort than writing a CUDA program.

But then also there are benefits of that and we will soon see that why the apparent difficulty is not significant deterrent, because in most cases, it is a 1 time job. So, somehow, it is not significantly popular to knife programmers. But if you discuss this with people, let us say working on embedded processing, and there is significant penetration from OpenCL programs. And also there are other vendors who are slowly realizing why OpenCL is important from the perspective of computer I mean pro computing with heterogeneous systems.

Another second important thing I would say if you notice that there are also FPGA based companies like Xilinx, who provide a lot of support for OpenCL. The idea is that is I mean, the way FPGA development works is you will write a program in a hardware description language like Verilog or VHDL. And there would be a synthesis tool, maybe a high level synthesis tool or a normal standard synthesis tool which will actually compile your structural or behavioral specification written in that high harder description language and create the corresponding LMT map for the FPGA.

But the thing is, it i mean when they say that, they are also supporting an OpenCL that would mean they provide alternate high levels in those tools through which if you are writing an OpenCL kernel, it is also compellable and runnable on FPGAs coming from Xilinx, so there is also an internet workload so as we can see that we have one common language through which if I write an assembly kernel, it can execute on any vendor supported CPU on any vendor supported GPU as well as a FPGA like device.

**(Refer Slide Time: 12:13)**



## Khronos Group Timeline

SIGGRAPH NEWS

| | | | |
|---|---|---|---|
| 1990's | Workhorse cross-platform professional 3D apps & gaming | OpenGL. | New Extensions to enable latest desktop graphics capabilities |
| 2000's | Ubiquitous mobile gaming & graphics apps | OpenGL ES. | OpenGL ES 3.2 released today to bring AEP functionality to core |
| 2005 | Safety Critical Graphics | OpenGL SC. | New Safety Critical Working Group - Call for Participation |
| 2008 | Heterogeneous parallel compute | OpenCL | OpenCL 2.0 specification update and C++ Headers released |
| Portable intermediate representation for graphics and parallel compute 2014 | | SPIR. | Provisional Spec Update and significant open source activity |
| High-efficiency GPU graphics and compute for performance critical apps 2015 | | Vulkan. | Adopted by Android ar platforms. Building ec |

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So this would be an alternative way to look at it. So in the 1990s, the workers cross platform professional 3d apps and games and gaming. That was OpenGL. And it came up with new extensions to enable desktop graphics capabilities. Coming to 2000s. We had ubiquitous mobile gaming and graphics apps being supported. So that actually meant OpenGL had an extension. OpenGL ES 3.2. It was released to bring this AP functionality to the court.

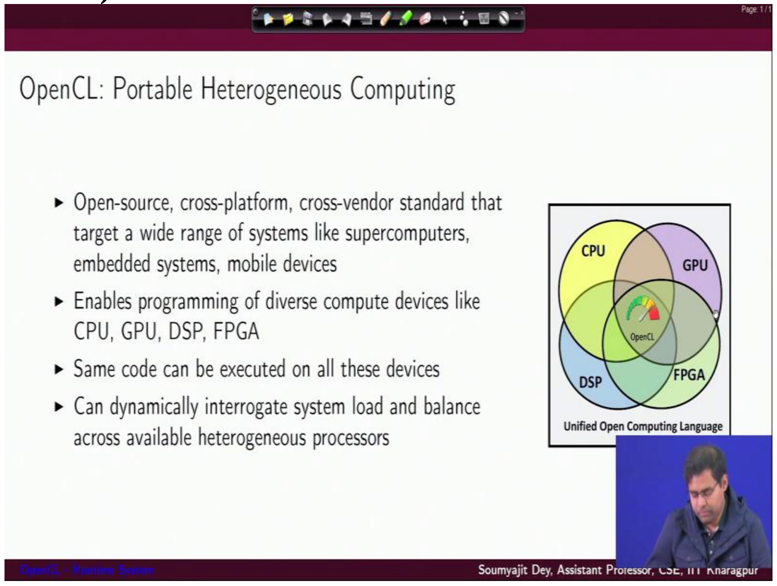
So they supported all these embedded domains like mobile gaming and graphics apps. And then we in around 2005, there was another extension of OpenGL into safety critical graphics, so that was the extension OpenGL SC. So it was a separate working group from called safety critical working group. And around 2008, there was this advent of OpenCL as a language for heterogeneous parallel compute.

And pretty soon there was this OpenCL 2.0 specification, and associated pro header files and libraries were released. I mean, the headers were decided, and the libraries, of course, had to be at released there is vendor backing. And around 2014, we have this portable intermediate representation from for graphics and parallel compute. And finally, something important happened around 2015, which was the creation of the Vulkan API.

So people found that, I mean, in the mobile world, for writing a program or writing a graph data parallel kernel is possible to write an OpenCL based heterogeneous kernel and run it. But maybe it is still not that popular from the mobile developer is point of view, and alternate was the Vulkan API, which is a very much OpenCL like language. And it is supported directly in the Android platform. So this would not mean you can have OpenCL code executing inside a mobile system.

For which you have to have OpenCL support on the mobile system, which is fairly common if you have an ARM molecule GPU or similarly other vendor supported GPUs who also provide the corresponding OpenCL ICD files. However, an alternative way, which is a bit easier would be write the code using the Vulkan API, which is also a very much OpenCL, or the original OpenCL like specification for writing such data parallel kernels, and they get direct support from the Android ecosystem. And that is also heavily used nowadays for writing performance critical graphical apps.

**(Refer Slide Time: 15:08)**



OpenCL: Portable Heterogeneous Computing

- Open-source, cross-platform, cross-vendor standard that target a wide range of systems like supercomputers, embedded systems, mobile devices
- Enables programming of diverse compute devices like CPU, GPU, DSP, FPGA
- Same code can be executed on all these devices
- Can dynamically interrogate system load and balance across available heterogeneous processors

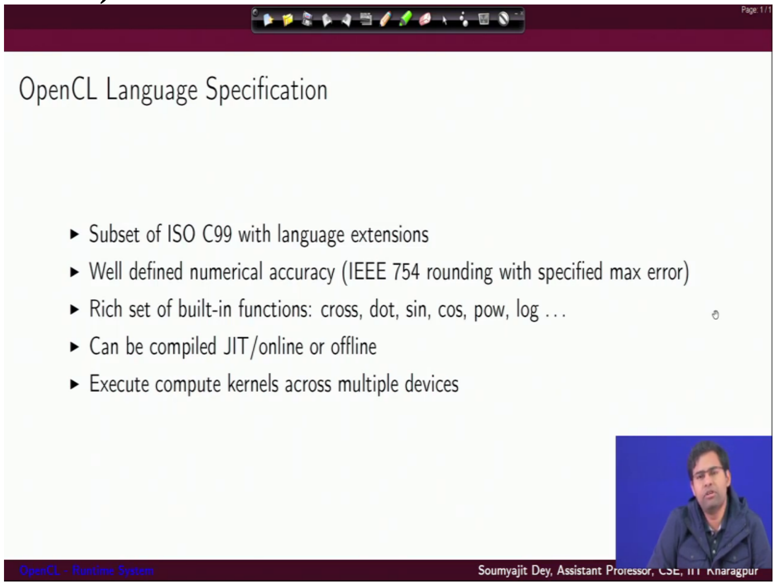
CPU
GPU
OpenCL
DSP
FPGA
Unified Open Computing Language
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, this is what we have been discussing that in a nutshell, it is a portable, heterogeneous computing language. It is open source, its cross platform cross vendor standard. It can target a wide range of systems starting from supercomputers to embedded platforms, mobile devices, etcetera using it, you can program diverse compute devices like CPUs, GPUs, DSPs, and FPGAs. And the same code can be executed on all these devices.

That is another very important point as we are trying to make that you write an OpenCL kernel. You can execute it in any of those devices provided the vendor of the device and gives you the support libraries. And it can also dynamically interrogate the system load and balance is execution across heterogeneous processors. Now this is something important we need to understand that if we are considering a heterogeneous system at any point of time there may be each of the constituent processing elements may be partially loaded.

So, from an idea load balancing point of view, when I submit an OpenCL kernel, it should be able to get an idea what is the amount of load in each of the compute elements and accordingly balances execution across the different platforms. And that is also a facility which is provided inside the OpenCL runtime system.

**(Refer Slide Time: 16:27)**

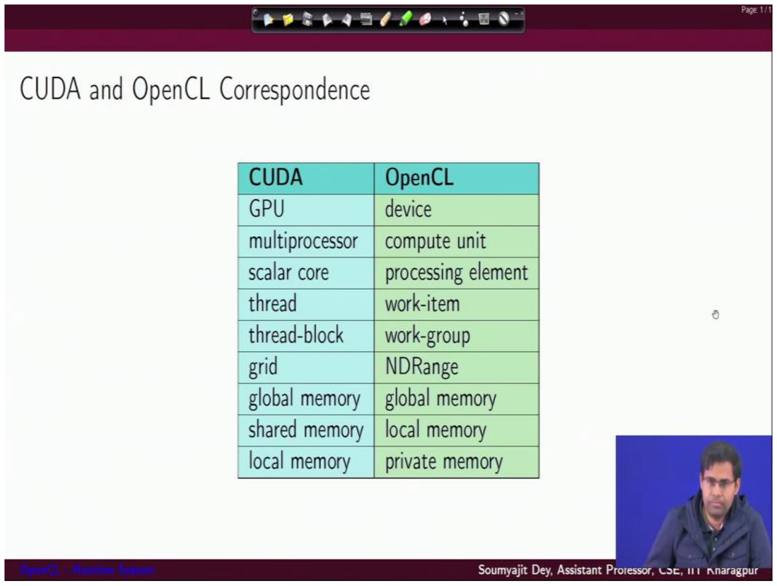


OpenCL Language Specification

- Subset of ISO C99 with language extensions
- Well defined numerical accuracy (IEEE 754 rounding with specified max error)
- Rich set of built-in functions: cross, dot, sin, cos, pow, log . . .
- Can be compiled JIT/online or offline
- Execute compute kernels across multiple devices

So, what about the language essentially, it is a subset of C99 with certain extensions, it has well defined numerical accuracy and of course, rich set of special transfer special math functions. And the usual ways that you can either decompile it or you can regenerate it have generate a binary in offline mode. So, ideally, I can have a runtime compilation and execution or I can generate offline binary ports are fine. Now, the last point which we were discussing earlier also that I can create a compute kernel and I can distribute it is threads across different devices.

**(Refer Slide Time: 17:12)**



## CUDA and OpenCL Correspondence

| CUDA | OpenCL |
|---|---|
| GPU | device |
| multiprocessor | compute unit |
| scalar core | processing element |
| thread | work-item |
| thread-block | work-group |
| grid | NDRange |
| global memory | global memory |
| shared memory | local memory |
| local memory | private memory |

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, here is something important since we are coming from a CUDA background. So, we will first need to understand how all the concepts of OpenCL are very much linked up with all the concepts of CUDA maybe some of the names get a bit changed. So, the moment, we are able to identify what is the correspondence, we will soon understand why they are so similar. The fundamental reason is both OpenCL and CUDA are built with the assumption of execution model as well as a memory model or I would say a platform model.

So if you remember how a GPU architecture was genetically define, we said that, it comprises a set of multiprocessors streaming multi processors for SMs is SM was containing shared memory L1 cash, and there were a set of special function units and there were a set of SM the course the scalar processor are espies. And these espies were actually used to execute the CUDA threads in with a basic scheduling unit of warps.

So we had a hierarchical arrangement of processors, the basic processors who are grouped together to create a Sims and all the essence together created the entire GPU, that is about the processing part. Regarding the memory organization part, we thought that, inside the GSM, we would have shared memory and will want cash. And of course, we also discussed that in final in modern GPUs, we also have the shared memory and L1 separate units.

But the other important thing was out, I mean, collection of such as stems together head into unified cash and then there was the global memory. And of course, for each thread we can have its own definition of local memory. Now, when we say local memory there would be a brand I mean 2 options finally, you have you will have local variables for thread which are private to each of the threads. They will be making use of the subset of register file which is allocated to each thread and our identity.

So, essentially from each threads point of view, it will get binding with an SP and a set of registers, is SP a memory scalar processor that this thread executes on this scalar process, it has access to this set subset of registers from the entire register file for storing its local variables. And if these histories are not enough, then it can further make use of local memory, which is essentially a small part of the original global memory.

So that was a memory model and execution model, OpenCL also considers a similar memory model and execution model. So, again the point would be that whenever any vendor is trying to create some silicon, which will be OpenCL compliant, here is to ensure that whatever process architecture it defines it should have a correspondence with this memory model and architectural execution model otherwise, it will be difficult to implement of corresponding libraries.

Now, coming back to this name correspondence, whatever in CUDA, we only had GPUs from Nvidia, whereas in OpenCL Sims is going to be a heterogeneous compute language supporting a set of devices we will call everything a device, every OpenCL device, a set of open cell devices together create a platform for computation. Inside the GPU, we had SM streaming multi processors in OpenCL will call them as compute units.

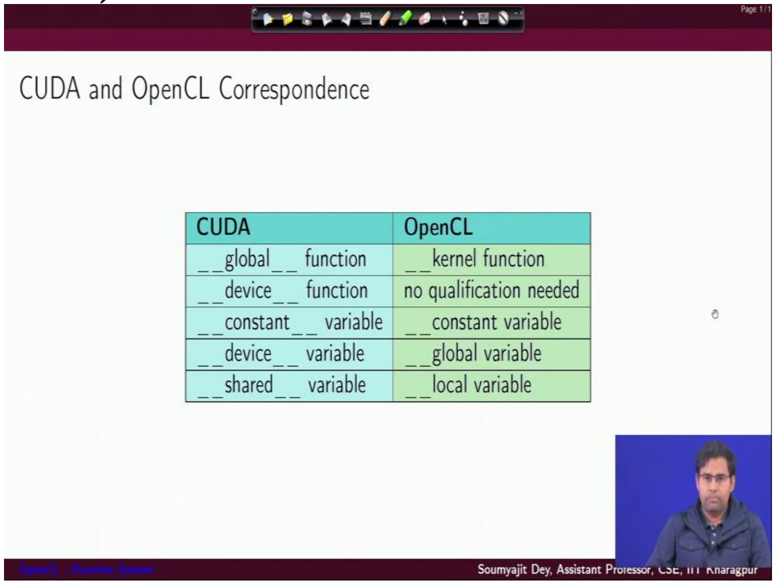
So we have OpenCL device and open cell device will contain ourselves compute units. Inside in each of the streaming multi processors of GPU, we had this collection of scalar core scalar process or espies. And here we will just call them espies or processing elements. So just to recollect a GPU is replaced by the term device because everything is an OpenCL device inside OpenCL device like GPUs, where we had streaming multiprocessors we have compute units in OpenCL semantics.

Inside streaming multi processors, we have scalar processors will now have processing elements. Inside the GPU, we had threads executing a collection of threads were executing threads will be called work items. A bunch of threads together was defining a thread block. A set of work items or set of thread-block work items will be defining what we call as a work group. So the block the thread block becomes what group the threads become work items.

And if you remember, we had a set of blocks, setup thread blocks, which together defined the entire grid of CUDA threads, the grid is known as ND range. So grid block thread that was the hierarchy here it is ND range work group and work item in OpenCL, the name global memory remains same, what was called shared memory in CUDA becomes local memory. That means, inside that each compute unit, we have a shared memory, which in OpenCL terminology would be called local memory.

So, it is local to that compute unit, but it is not local to the processing elements. So, it is local to the compute unit. And the original definition of local memory would be called as private memory in OpenCL.

**(Refer Slide Time: 22:45)**



## CUDA and OpenCL Correspondence

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

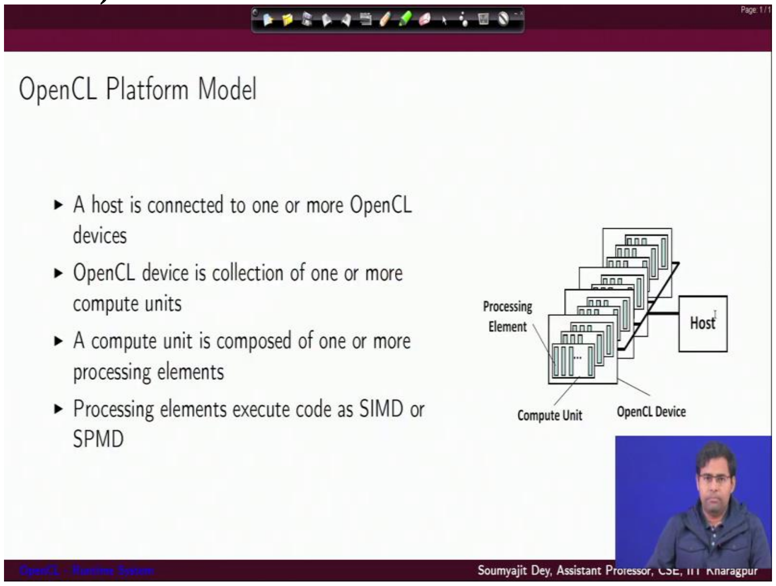
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, in terms of functions, if you remember for functions, we have the following keywords in CUDA. So we had global device. So whatever was a global function becomes the name of kernel function. So a global was a function which was scalable from the host but executable on the device in CUDA, here we are calling it a kernel. So it is OpenCL kernel we execute on a device and a device function means it is scalable as well as executable only in the device. In case of OpenCL such a kernel does not require any qualification.

Now, coming to variable types, we had constant variable types that name remains same in OpenCL. So that would mean a variable which would be only defined once it would encourage exclusive storage space will be the constant memory segment in the GPU. Now that terminology and idea would remain same. Then in CUDA, we had this keyword called device for variables, which would be defined in the GPU GB RAM.

In case of OpenCL will have the device on memory where it will be defined as a device on global memory where this will be defined with that keyword global? Again since the shared memory the name shared memory is being used with as a local memory. So, the shared keyword for this CUDA variable gets replaced with the local keyword. So, shared is local and device becomes global in terms of variable names.

**(Refer Slide Time: 24:17)**



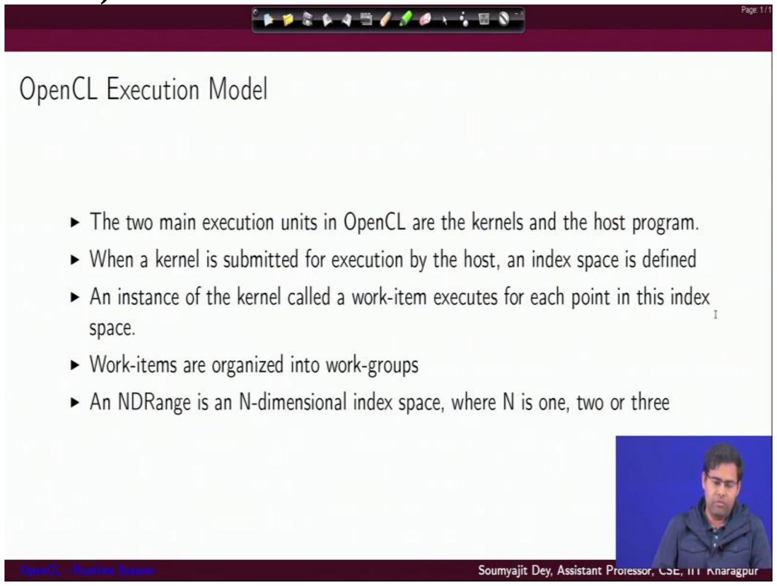
## OpenCL Platform Model

- A host is connected to one or more OpenCL devices
- OpenCL device is collection of one or more compute units
- A compute unit is composed of one or more processing elements
- Processing elements execute code as SIMD or SPMD

So, here we have a pictorial view of the OpenCL platform model. So, this is how the OpenCL platform model is conceived. And as I was saying, any vendor who is trying to create an OpenCL compliant processor data parallel processor has to conform to this theoretical model. So we identify that there will be a host device, which will connect to 1 or more OpenCL devices and as an OpenCL devices as we have discussed earlier, it is a collection of 1 or more compute units.

Each compute in it will have one or more processing elements. And these processing elements will execute code in a data parallel way. So, or SIMB or SPMD, that single program multiple data or like, that is the terminology they will use.

**(Refer Slide Time: 25:09)**



OpenCL Execution Model

- The two main execution units in OpenCL are the kernels and the host program.
- When a kernel is submitted for execution by the host, an index space is defined
- An instance of the kernel called a work-item executes for each point in this index space.
- Work-items are organized into work-groups
- An NDRange is an N-dimensional index space, where N is one, two or three

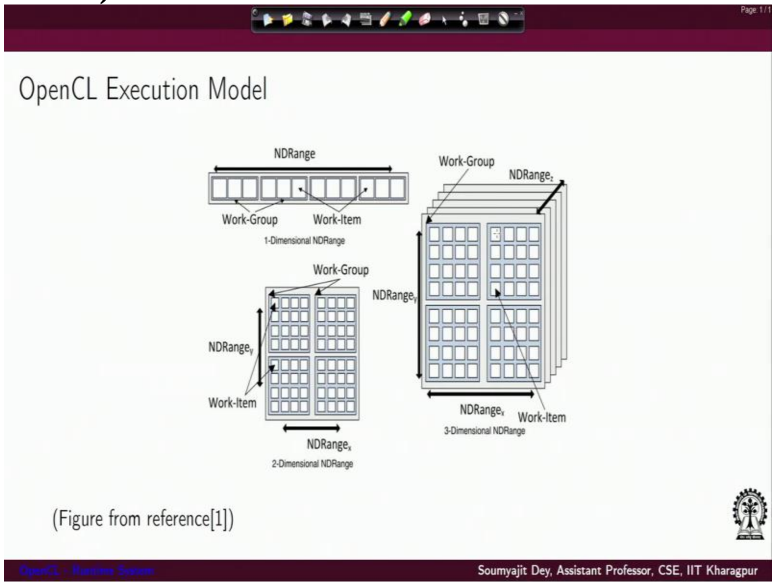
Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, similar to CUDA programming OpenCL also we have 2 main execution units, they are the kernels and the host programs essentially the host program will be occupying this host device and it will be orchestrating the execution of the OpenCL kernels in the different other open cell devices. So, when you submit OpenCL kernel using a host program to 1 of the open cell devices, it defines an index space. An instance of the kernel is I mean is a work item and it executes for each point in this index space.

So, if you remember in CUDA, we had agreed and installed that we launched outset of thread blocks. So, if you see here that is now the index range. And inside this we are instead of launching threads, we are launching this work items, each work item will execute and try to compute some value for one of the points in this index space. Now, the work items are now organized into work groups as we are discussing earlier.

And by ND range what we mean is that in the N-dimensional index space, where the value of N can be 1, 2 or 3, so, in the highest possible dimension, we can have a 3 dimensional arrangement of this OpenCL work items.

**(Refer Slide Time: 26:37)**



## OpenCL Execution Model

(Figure from reference[1])

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, we I mean this picture is kind of reproducing us the idea of parallel execution of threads and the different OpenCL keyword concepts. If you remember our initial introduction to the CUDA program, land programming language, we showed the arrangement of CUDA threads. Where the threads are bunched into thread blocks, the blocks are bunched together into a grid. Here we have a similar thing, we are trying to show different possible cases of one dimensional 2 dimensional as well as 3 dimensional element of the index space in the ND inch.

So, we have a 1 dimensional ND range where we have multiple work groups, each of the work groups are containing work items. Similarly, you can have can have a 2 dimensional ND range, which will contain 2 dimensional work groups and the work groups again will contain the work items arranged in 2 dimensions. And similarly, I can have also a 2 dimensional indie ranch where the work groups are also packed into dimensions. So with this basic introduction to the OpenCL execution model, will end this lecture. And in the next lecture, we will be continuing the discussions further. Thank you.