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

Module No # 03
Lecture No # 14
Multi-dimensional mapping of dataspace; Synchronization (Contd.)

Hi, so welcome to the lecture on multi-dimensional mapping part 2 so in the last lecture we have been discussing about how to compute for a given thread how to compute the global thread id how to compute of course how to compute first compute its block id this is position of in terms of the exact block where it is it. Then the position of the thread inside this block and then use this block number and position of the thread values to compute the global thread id over that is the exact position or index of the thread among the entire packing of threads that is there right.

(Refer Slide Time: 01:07)

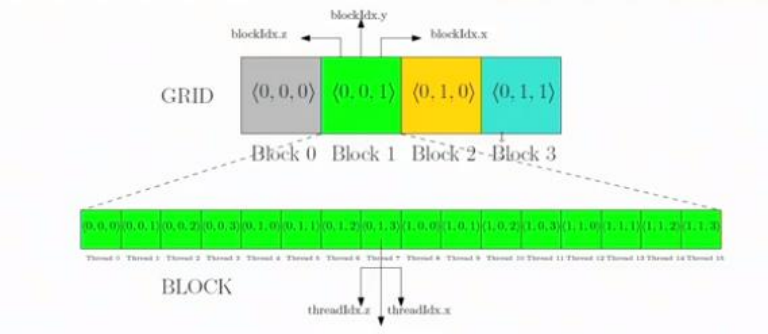


Figure: Global Thread IDs

So this was the example we have been using just to recall what we are really trying to do.

(Refer Slide Time: 01:18)

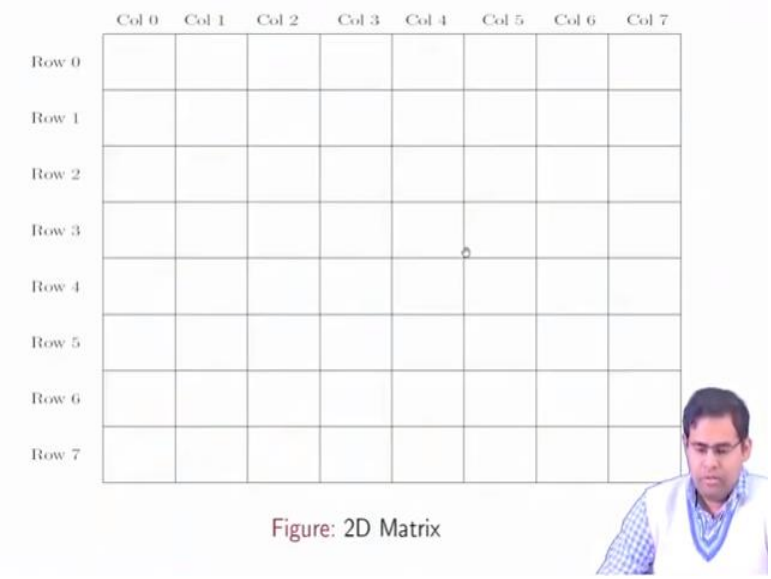
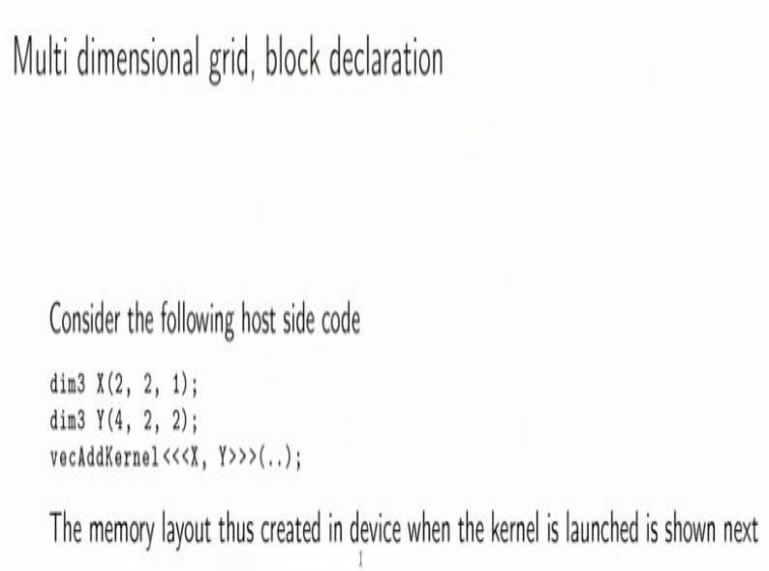


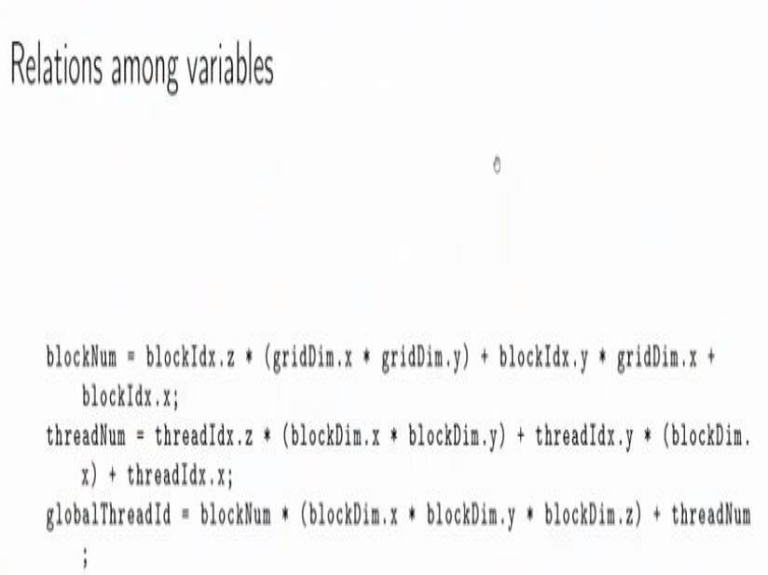
Figure: 2D Matrix

(Refer Slide Time: 01:27)



We are trying to here design a 2D matrix space and for some reason we are trying to define this 2D matrix space using a collection of threads which are arranged in 3 dimension with respect to the thread block and we have 4 blocks arranged in 2 dimensions. Here in our example we have colored the blocks in different ways.

(Refer Slide Time: 01:42)



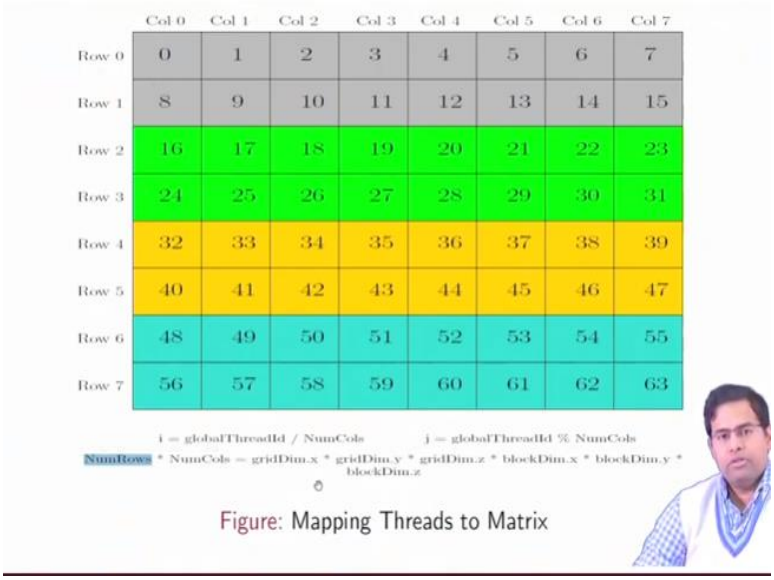
And now we can understand that given this relations each thread can compute its exact position and then so this is the relation among the variables that we have already discussed and so inside your program you can have this kind of program statements which a compute thing thread that is the CUDA kernel can figure out using this kind of program statements that what is the global id of the thread right.

So what is very important here is when the kernel is launched and it is executing you have as we have discussed earlier you have single instruction multiple thread right the (()) (02:22) model of computation. So that is same piece of code that is the kernel and that piece of code is getting executed by all the threads whose packing has been decided by the launch parameters right. So using this kind of expression every thread can compute a unique global thread id.

This is the most important thing with respect to distinguishing the thread with respect to other threads. For every thread I have the unique combination of block id's and thread id's using which it can compute a unique global thread id. Now why this is very important because this is the very parallel program right. So every thread needs to find out what is the unique work it is going to do that is what is the unit position in the memory from which it is going to fetch operands what is the unique positions the memory where it is going to stored operands based on whatever computation it is going to do that is specified in the core.

So for getting this unique positions in the memory or in terms of some position in a specific data structure I would say for with respect to the programing (()) (03:33) the threads need the unique id to be computed and that is basically the global thread id.

(Refer Slide Time: 03:39)



So as we can see if we now apply a earlier discussion of threads and blocks as we have been discussing that we are coloring the blocks in 4 different colors and each blocks is now containing this 16 threads then if we map a 2D matrix into this collections of threads and blocks then this is how they would look like. So as you can see I have this 4 blocks so of course this is the 2D picture but technically as we know that since CUDA this is the programming C based programming language and the memory access (()) (04:15) as to be extremely sequential so finally the arrangement of this locations in this memory is also sequential.

So this locations would be extremely sequentially up from 0 to 3 64 locations you have the first 16 locations would be consumed by block 0 that means all the threads with those id's mapping t block 0 and next 16 to block 1, block 2 and block 3 right. So for every thread here I can compute suppose I am trying to do some matrix operation so then the thread as to figure out using its combination of thread id and block id as well as grid dimension block dimension variables that what is the corresponding position which it is going to work with respect to the matrix arrangement right.

That means it has to figure out what is the row column index combination on which for which it really corresponds we are assuming here that we have launched 64 threads each of the threads are going to do something about a unique locations of the matrix. So let us say we are trying to say that the thread which is of id 18 is going to do something about the data located at the second row and the second column right.

First of all threads the need to be map itself back to this position that yes I am thread who is suppose to do something about the data located at the second row and second column position which would mean it has to figure out the corresponding this values right. The i and j values that is the row and column values here right. So how can it do that first of all the thread would use the expression which have divided discussed earlier there how are thread can really compute its block number?

How are thread compute its thread number? It can use this expressions to deliver what is global thread id that means what is its exact position among all the corrections of threads right. So using those relations like this thread should be first able to compute this value 18 right that this among this global position which are ranging from 0 to 64 I am the 18 thread. Now it has to use this number and the other values that is the matrix dimension values to compute okay what is the row and value column on which I am going to work.

So that it can figure out just by dividing the global thread id with number of columns so it knows that okay I have covered so if I just divide it by number of columns then definitely I would get that I correspond to this they can row here right. And then again if I do a percentile operation that would give me that what is the column at which I am located which is obvious right because as we can see that this percentile with 8 is giving me 0.

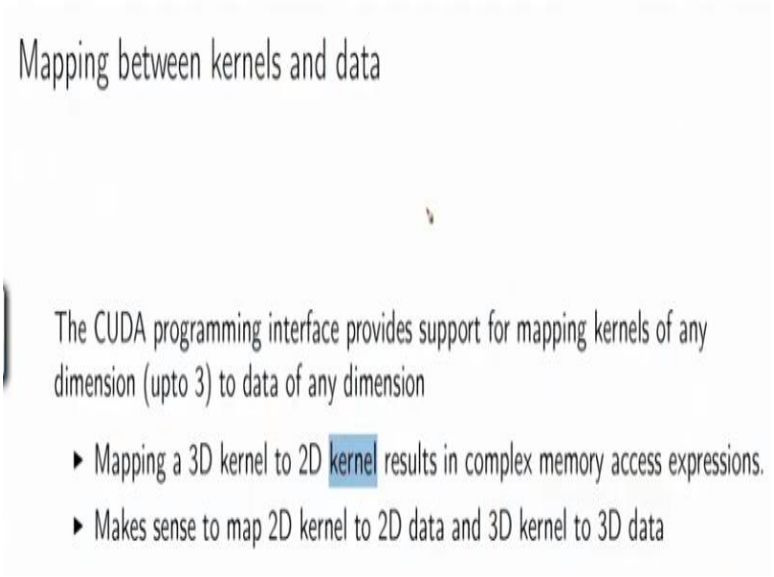
So I am located in this column 0, column 1, column 2 like that right so in this way I can keep on doing the percentile as you can figure out what column I am corresponding to I can divide by the total number of columns and I can find out what is the row and I am corresponding to right. So for 18 I will be able to apply this method and I should 18 divide by 8 that and of course we are not worried about the reminder so that gives me a question of 2 here so i know that the row

number is 2 and of course 18 percentile here with 8 that would give me reminder of 2 so I know that the column number is also 2.

So in that way the thread is able to discover which is the location in the matrix for which it is going to do something over right. Now of course we can see since this is that there is 2D arrangement another relationship should also hold that i have the total number threads which is given by this expression that the dimension of the entire grid that is the x, y, z that is the total number of blocks multiplied by so this part gives me the total number of blocks of course the dimension of grid in the x, y and z directions.

Multiplied by the total number of blocks inside the thread that is the block dimension x, y and z dimension this gives me the total number of threads which should also be equal to that total number of rows multiplied by total number of columns. So overall I have 2 different ways to find out what is the total number of threads in this case.

(Refer Slide Time: 09:01)



Now in general so this is just an example situation we use this kind of a 2D representation of the data and a matrix example and we use this definition of blocks and grid to figure out that okay if I launch the threads using this kind of launch parameters then with respect to the actual data collection which may be in 2D space how can each thread compute what is the data location for which it is going to do some work right.

But typically how would you like to go about it so the CUDA programming interface is going to provide you support for mapping kernel in any dimensions to data of any dimension right. You can but of course but of course you are limited by the things that your kernels the number of blocks as to be limited to 3 dimensions and inside the block your thread packing has also got limited to 3 dimension.

But inside this I can always vary 0, 1 and 2 I mean I can only have the x dimension I can have x and y 2D I mean I can have 2D packing of blocks with 1D packing of threads 1D packing of blocks 2D packing of threads all those combinations can come in. But how do you really want to pack the indexes depends on your choice of the problem. Suppose you are trying to map 3D kernel but I mean you are trying to map you are trying to map a 2D data space using a 3D kernel that is a bad decision to make.

Because then what will happen is your memory access expression are going to be complex right however it makes sense that if you have a 2D data space the actual data you are talking about may be it is a 2 dimensional matrix then you set your launch parameters in such a way that your kernel is fundamentally 2D that means you have 2 dimensional blocks and also 2 dimensional arrangement of blocks in the grid. Similarly if your data structures is 3D it may help to design your expression of the memory which respect to the global thread id in such a way that you have 3D kernel which is mapping nicely the 3D data right.

(Refer Slide Time: 12:03)

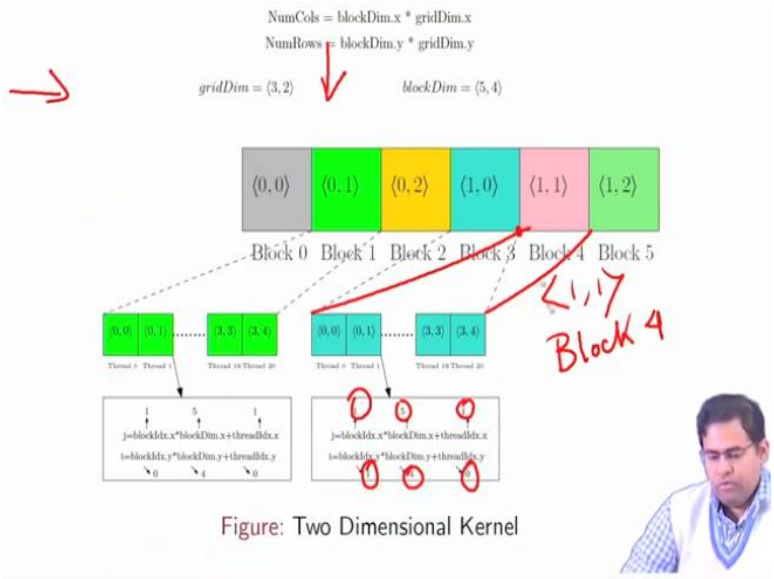


Figure: Two Dimensional Kernel

Okay this is the small correction here that we have now proceeding further so in this 2 dimensional kernel example will take another different case here that suppose we have already seen that what should be (()) (12:16) guide line we are talking about really mapping in 2 dimensional space we should be using 2 dimensional kernel definitions right. So let us define a grid which is 2 dimensional which is the dimension of the grid right that is you are talking about in x dimension we have the block id x changing from 0, 1 to 2 and y dimensional you are going to have block id values that is block id x dot y values 0 to 1.

And inside this block you have 2D packing of in total of 20 threads 5 times 4 then this is how your blocks would be arranged I mean again we are using a color coding to kind of represent the different blocks. So I am going to 6 block the issue is see how things change with this dimensional coding since the grid is defined with index 3, 2 so your block id is going to change from 0, 0 upto 1, 2 right.

So you have 00, 01, 02 like this since you are going to spread the block id x dot x variable from 0 to the index 2 and then again you would be increasing the value in the y direction and then again the block id x dot y would be 1 and again your block id x value would range from 0 to 2 right. So in that you have got this arrangements of 6 block now if you pick up the one example block for example this block 1 as we can see that since the block dimensions have been defined here as 5, 4 so its total 20 threads this 5, 4 will actually define how the threads are going to be indexed right.

So since this 5, 4 the threads would be indexed from 00, 01 like this upto 3, 4 so here this is thread id x dot this is the basically the x dimension this is the y dimension since this is the x dimension so we understand the thread id x dot variable should range from 0 to 4 and since this is the y dimension the value is 4. So thread id x dot y is going to range from 0 to 3 right so overall I would have thread 00, thread 01 and the final thread with values 3, 4 in total we have 20 threads I hope this is clear.

So again I would repeat so the x comes dimension comes first then the y dimension whereas when I am just writing them in order the x dimension is increasing first and then the y dimension is decreasing. I mean that is how you would like to remember it here so now if I am looking for a

specific thread here let us this thread so again let us do a recollection of what would be an thread id and other values.

So since this is the 2 dimensional kernel I am doing a 2 dimensional kernel mapping to 2 dimensional data space. I am trying to pick over the I and j values right that is the row and column in the index values here. So the j value is given by the block id x dot x times block dimension dot x times plus the thread id in the x direction right. So what are those individual values so as you can see the block id for this is basically 0, 1 so this is the block id x dot x value this 1 which is so here right.

This is the block id x dot value this is not required here the block dimension is again in the x direction it is 5 so this is the block dimension so this is 5 here the block id x dot x value is 1 because of this right and what about the thread id x dot x value. So the thread id x dot x value is 1 because here this is 1 and thread id x dot y is 0 which is not again required here. So this is how it is right so using this i can compute what is this j index?

So that gives me the column index here right similarly I can also compute the row index you continuing a 2 dimensional data layout here.

(Refer Slide Time: 16:46)

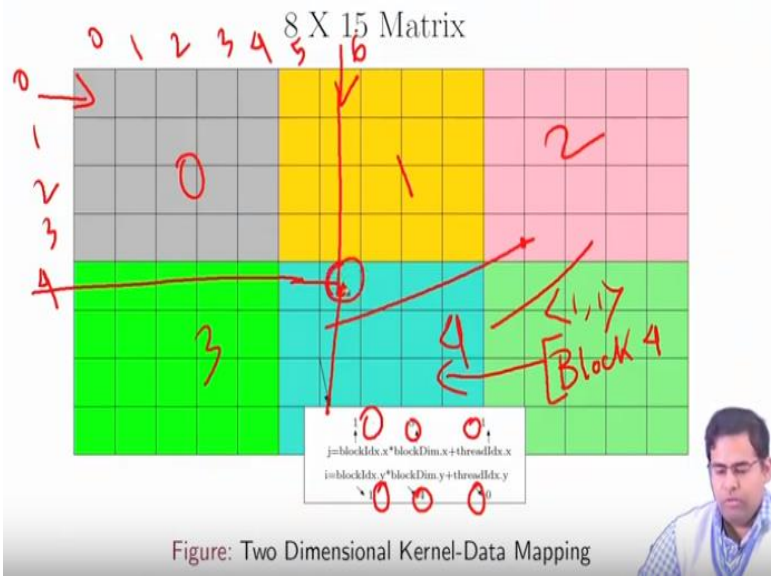


Figure: Two Dimensional Kernel-Data Mapping

So coming back here the column in so fine here if we just check it for this specific example only 151 so if you just compute here what do you really get so you have what this. Since that is 6 and

also let us do the same for the I value as you can see the block id x dot y not that is we are talking about block 1 here. So block id x dot y is 0 this 0 so in the block dimension in the y direction is this right plus the thread id x in the y direction.

So what is the thread id x in the y direction here so again you have 0 here this is 0 so that is essentially I am sorry so this would map to this I0 the first row and then what we computed here earlier was sorry yeah. So here I was computed as 0 and this was computed as 6 so your column index would be 0, 1, 2, 3, 4, 5, 6 so that would be here right that is it. Now coming back to for our example if we pick up some alternate values.

For example let us pick the some entity from block 3 here so block 3 as you can see index here so block 0, 1, 2 and then 3 so the block id x dot y is now becoming 1 small correction just make 2 on the figure here. So ideally since so the block id x is also 1 and the block id x dot y is also 1 essentially we are speaking of here this figure block 4 right. So then inside block 4 if I pick up for a specific thread there is a first thread a thread id is I mean the thread 1.

So for the thread 1 inside block 4 as you can see the thread id starts from the for the zeroth thread is thread 00, 00 and thread 1 is 01. So the thread id x dot x is 1 thread id x dot y is 0 and the since this is the fourth block so I mean among the zeroth first second then third block is 10 forth block is 11. So I have the block id x dot x and block id x dot y both as 1 and then I have the block dimension here right and those we have already defined earlier in the x block dimension is 5 and block dimension dot y is 4 right.

So that is what you have so with this finally I can say that okay here what is the position I am talking about so would give me we have the column position and that is you have 1 times 5 + 1 that is 6 and so the sixth column and which road you have is the fourth row right 1 times 4 and that is what we have right. So row 0, 1, 2, 3, 4 so you have the fourth row and then so in terms of the column is sixth.

So 0, 1, 2, 3, 4, 5, 6 right so it should be this position here so as you can see you have this blocks right so you have you are not talking about the fifth block so the block id’s are 1 comma 1 and inside this block you have the block dimension coming to 3 this block dimension for each block

you have the block dimension which is 5. So 1 times 5 + 1 so that is your j and if we speak about the block id here in terms of the rose.

So again you are moving block dimension in the y dimension when you are computing the row so you are in block id x dot y 1 that would automatically got multiplied by 4 so then you are already you have already covered the first rows and then you have your thread id is 0 so essentially you will be in the first row in the fourth block here that is essentially if you compute the rows from 0 zeroth was second, third, fourth so this is the position you would be talking about yeah.

So just ignore this arrow so if you are talking about block which is the forth block so this is the zeroth block first block, second block, third block, fourth block even by 1, 1 and for this block I have got the thread id x values dot x as 1 thread id x dot y as 0 and with this I am able to compute this is the position where I am talking about right. This is the location I am talking about so in this way you can see that if you are trying to define for a 2D matrix you are trying to use definition of grids and blocks in the 2D.

That means you have a 2 dimensional packing of blocks and you also have a 2 dimensional packing of the threads inside the block so this is the 2 dimensional packing of blocks this is the 2 dimensional packing of grids inside the block it nicely maps here to specific locations. And now let us take an example of a 3 dimensional mapping of a kernel on a 2 dimensional data space. So since we are trying to talk about a 3 dimensional making but again I would again repeat some simple thing that at all we went for this because of initial hypothesis was that a 3D mapping a 3D kernel maps nicely to a 3 data space and 2D kernel maps nicely to a 2D data space.

So since as we can see that our hypothesis is quite validated because now the excess expression since for this thread which is with the block id 1, 1 and thread id 1, 0 I am able to execute this 2 expression and compute a corresponding I and j values and does not the expression look really nice and simple. Because the j value which gives me the column index I have the very nice expression because block id multiplied by block dimension plus thread id everything is x axis right.

Similarly for the row index block id multiplied by block dimension plus the thread id again everything in the y axis right. So this feature as you can see it is quite intuitive and it is easy to map because since I have 2D arrangement of threads and the threads are going to compute on a 2 dimensional data space that inside the data structure definition that itself is a 2 dimensional arrangements.

So once I map the collection of threads in 2 dimension the access expression for the row and the column indexes automatically map dimension wise. This is the most important thing that we really caring about here. Since the data structure is having a specific dimensional arrangement and we are packing the thread in an identical dimension arrangement I have got this kind of very simple expressions which only involves the x axis variables or only involve in the y axis variables right. Now we will be soon seen that this idea maps for more complex data types also

(Refer Slide Time: 28:02)

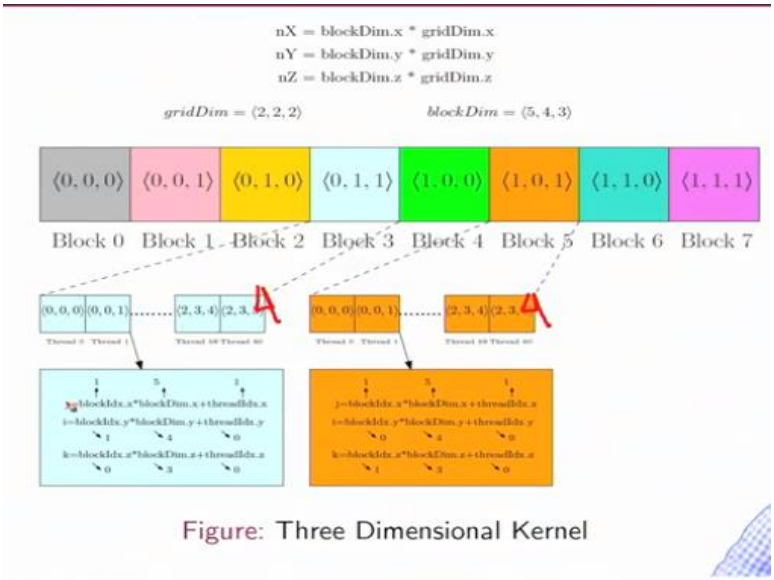


Figure: Three Dimensional Kernel

For example let us talk about the 3 dimensional kernel which is going to work on a 3D matrix. So for this 3 dimensional kernel let us consider 8 blocks so your grid dimensional is 2, 2 arrange in 3D and you have each block containing 60 threads they are mapped in 3 dimensional as 5 cross 4 cross 3 and so we are having this 8 blocks with their indexes called just like we have the earlier examples.

If you pick up 1 block inside that again you have this collection of 60 threads with id's all 0 to 2, 3, 5 why because you have the last thread we will have 2, 1 less than this one less than this once

less than this. So this should be 4 sorry here so this should be 4 let me just put a correction here so last index have to be 4 and so again when I am trying to compute let us pick up 1 thread for this thread when I am trying to compute that exactly in which location in the 3D matrix as this thread corresponds to I can find out the i, j and k because now the 3D matrix the positions for the location in the 3D matrix corresponding to this thread.

I am saying that okay this thread will compute something for this location in the 3D memory again the (()) (29:37) are all in x direction block id x times dot x times block dimension dot x plus thread id x dot x similarly all in y dimension all in z dimension right yeah.

(Refer Slide Time: 30:06)

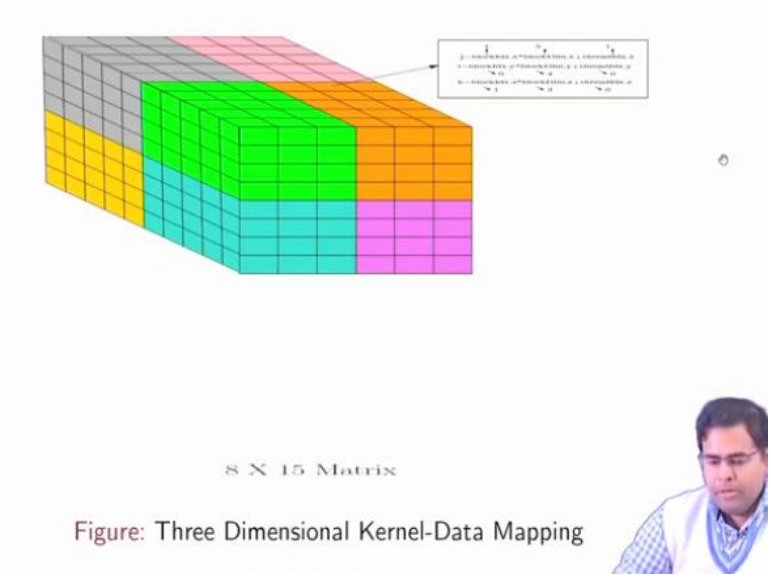


Figure: Three Dimensional Kernel-Data Mapping

So this is how the arrangement to look for the 3D collection here so you are having 8 blocks now you have 2 dimension definition of 2 in the x axis I mean you have in the x side you have blocks in the dimension in x size the value ranging from 0 to 1 so it is 2 similarly the y axis and similarly x axis so you have in total 8 blocks and inside each block as we define you have got 5 threads 4 threads for the y dimension and 3 for the z dimension.

So you can see you have such arrangement here 1 dimension is a collection of 5 in 1 dimension it is a collection of 3 and in 1 other dimension it is the collection of 4. So this is how you can say that a collection of threads map to a 2D or 3D data space in general I can have for a complex application I can have a more complex data structure considering high dimensional data and I can look at dimensionality of the data and accordingly I should be able to decide a suitable

dimension of my blocks how to pack the threads that the blocks in a multidimensional way how to pack the blocks inside the threads again in a proper multi-dimensional way.

So that I can design a simple expression I can access the location inside the actual data structure and compute using the threads we will see more into that in future that how this can also create some problems with respect parallel computation and collaboration among multiple threads for now let us stop here thank you for this (()) (31:58).