Occupancy:

In this video we are going to discuss about one of the key concepts in CUDA programming called occupancy. Occupancy is the ratio of active warps to allowed maximum number of warps per SM and it can be calculated using following equation. In a given CUDA core, instructions are executed sequentially. When one warp stalls because of the latency in arithmetic or memory operation, that SM switches context to another eligible warp. So ideally if one warp stalls, we want to have enough number of warps to keep the cores occupied. So, if the occupancy of a particular kernel is a high value that means even though one warp stalls execution there will be other eligible warp so that of cores always will be busy. In this equation, term max number of warps in SM can be found out by either looking at by device microarchitecture's documentation or by querying the device with max threads per SM property and dividing it by warp size which is 32. On the other hand, active warp counts per SM depends on available resources in device and resource consumption of your kernel. So let me show you an example calculation of this value. let's assume a kernel uses 48 register per thread and 4096 bytes of shared memory per block. Here there are two factors which affect the number of active warp in a SM. Register usage and shared memory usage. So here is what we are going to do now. I am going to calculate allowed warp count considering these resource separately and minimum of those two values will be the one which limits the active warps per SM. So let's first find the allowed active warps considering the register usage of our threads. Now here, one thread uses 48 registers. So for a given warp, register usage will be 48 multiplied by threads per warp which is 32, which equals to 1536. And if I consider GTX 970 as the device I run this kernel, then we have total of 65536 registers per SM. So we can divide total register value per SM by by register consumption of our single warp to find out allowed active warps So that value will 65536/1536 = 42.6. But this is not our last value. There is a concept called warp allocation granularity value. In SM, usually depending on available number of warp schedulers, warp allocation happen in group wise. For example, if our device has warp allocation granularity value 4, this means warp allocation happen in group of 4 warps. We might only need 1 warp to execute particular kernel. Still SM will allocate 4 warps as its granularity is 4. Now I have GTX 970 device here. And it has 4 as warp allocation granularity. So SM can allocate warps count which are multiplier of value 4 only. Therefore in this case allowed warp based on register usage will be 40. Now let's calculate allowed warp count based on the shared memory usage. GTX 970 has 98304 bytes of shared memory per SM. And these memory will be shared by all the thread blocks in the SM. Here in our kernel we used 4096 bytes of shared memory for block. So based on this values allowed thread block count will be 98304/4096 = 24. And if we consider 128 as our thread block size, which means 4 warps per block So based on shared memory we can have 96 warps per SM. But remember for device with compute capability 5.2, we can have only 64 warps at max per SM. So this means our active warps count per SM does not limited by shared memory usage. So here register usage limits allowed active warps and that value is 40. So our occupancy will be 40/64 which is 63% of occupancy. Knowing how to calculate the occupancy value is very important to get the grid on the occupancy concept. But CUDA toolkit provide tool called CUDA occupancy calculator which does this calculation quite easily. So let me show you how to use CUDA occupancy calculator now. This tool is actually a excel sheet. This excel sheet has 4 tabs. Calculator tab which we are going to use in a minute, help tab which explain how to use occupancy calculator GPU data tab which contain device query information for device with different compute capabilities and of course copyright tab. In the calculator tab there are different section highlighted in the different colors, and there are some graph which highlights the co-relation between occupancy and varying block size, shared memory usage per block and register per thread for given configuration. In this green section we can set details of our device. So first select compute capability of your device. When you select compute capability, below gray area, This area will be populated with device information. There you can see the physical limits for device with selected compute capabilities. Second information you have to put here is the shared memory size which you can select from the drop down box or you can fill it manually by looking at the device limitation section. As 3rd parameter you can provide cache mode for execution. For now you can leave it behind and move on to the next section. We will learn about this parameter when we are discussing about caching in CUDA. Then in the orange section we have to specify the execution configuration and memory usage of our kernel. You can fill the threads per block based on kernel launch parameters. To find out the registers per thread and shared memory per block you can compile your program in nvcc with ptxas options which will output the information about registers, local memory, shared memory, and constant memory usage for each kernel in given CUDA file. For example here I have program called occupancy test, in this occupancy test kernel, I am using 8 integer variables X1 - X8 and those together and store the results in to the global memory. Now let me compile this program with above given options to nvcc compiler. So go to the location where you have your CUDA file and run the command shown in here. So the outcome will look like this. In this output, in the last line you can see the register usage of reach thread for our kernel. And if there is shared memory usage then that will appear here as well. As you will see in upcoming videos we can allocate shared memory dynamically as well. If so, you have to explicitly add those values to the shared memory as well. So, if I add these values to the occupancy calculator, you can see GPU occupancy data in the blue colored section. Now change these values a bit and observe the change in outcome when we change the device configuration input or resource usage per kernel. Also down below there is a yellow section which highlights what we obtained from our manual calculation. Here you can see that maximum thread blocks per multiprocessor has been limited by registers per multiprocessor and that warp value is 40 as we calculated before. Also here you can see that calculated occupancy value which in this case 63%. for more information on how to use occupancy calculator you can refer the help tab and it includes all the information I just present to you. But keep in mind that higher occupancy does not necessarily means higher performance. If the kernel is not bandwidth bound or computation bound then increase in occupancy will necessarily increase the performance. in-fact making changes to increase occupancy can have other effects such as additional instructions more register spills to local memory which is an off-chip memory and more divergent branches. As with any optimization you should experiment to see how changes affect the wall clock time of the kernel execution. For bandwidth bound applications most probably increase in occupancy can help to hide the latency of memory access. And therefore improve performance. Ok, with all these knowledge, let me present you few basic point that would help your application to scale better in GPU. Keep the number of threads per block, a multiplier of warp size. Avoid small block size as too few threads per block lead to hardware limit on number of warps per SM to be reached before all the recourses are fully utilized. So start at least with 128 threads per block. Adjust block size up and down according to the kernel resource requirements. Keep the number of blocks much greater than the number of SMs to expose sufficient parallelism to your device. Conduct experiment to discover the best execution configuration and resource usage. Ok now you know how occupancy is for performance of our kernel. But keep in mind that occupancy is not the only factor effecting the performance and after a certain level of occupancy reached further increase may not lead to performance improvement. As you progress with this course you will learn these factors as well.