Reduction with dynamic parallelism:

In this video, we are going to implement a reduction algorithm with dynamic batteries. You see how the Falu we use to sum up one trade block in interleaved approach here. In each iteration, we sum up the value of it. Away from the given trading. We said I will start off that value from half of the block size and reduce it by half in each iteration. If the trade block size is 16 in the trades which perform summation in each iteration can be visualized in this way. In first iteration, only four Hatred's would perform the summation in iteration on the first footrests, perform summation and in the last iteration only first performed the summation. Now this pattern closely resembles the kernel launch pattern we shown in the dynamic parallelism. So here's what we are going to do. We are going to replace the loop in the reduction with interlude approach with recursive tailgunner launches. So let's implement this approach not. Of a camel signature would look like this. We need to have input and I point to output point and size of the trigger for this kernel as arguments. Now we need to convert the global data point to local point for this book. We can do this by using look index value as offset as we did in the previous occasions. Then we have to stop condition here of a stop condition is the block with size to it so we can sum up two elements in the block indice rooted in the block. So let's do that now. Now we outperform in police reduction before the invocation of the child rate, so let's add up the values correspondent Carondelet Index and offset away from that index here. The offset is half of block size. So we need to calculate that first and then some abomination to values. Then we need to wait until all the trends in the block complete this step before moving on to the next step. So let's add secret function call here. This function call will guarantee that all the threads in the book reach this point before any of the trades moving forward. No, we can perform recursive challenges, so in the first grade in Esterbrook, we will launch a hybrid with half of the block size or stright amount of plates. And remember to pass the localized data point as the child would ask for it, then we have to wait until all the child would complete the execution. So we need to devise synchronized function call here, as we did in the Hosken launches. This function call will block the thread with the ID value zero execute in further instructions. Also, we need all the threads in the block to wait until thread zero to finish. So here we need another central function. Call again to perform Block Weitzen organization. That's it for our Gernhardt main function for this program is very similar to what we had in the interleaved implementation. The only difference being the conolly we launch here. To be honest, this is very inefficient program. So let me compile this and run it for smaller input datasets, then 128 megabytes we used before. So let's get our input size to two to the power, 22 or 16 megabytes of data. OK, let's compile this program. Not here. Also, I have to say that they are S.H. and our RDC compiler options. And after the compilation, we can run this program. So as you can see, our kernel is a valid one as both C.P.U and Deepu implementations. IEL, same with us. But keep implementation take way more time than keep your implementation. Part of this due to the unnecessary synchronization in economy, when a child is involved, it's we, Wolf, Memrise, fully consistent with parent because each child only needs its parent value to conduct the partial reduction. The inbuilt synchronization performed before the child gets are, which is unnecessary. So let's remove all the synchronization operation we perform in the previous gonner and then we can observe the performance improvement of this program. OK, let's compile this program again and write. OK, this is our output and it is a valid one, but also you can see that is still our implementation take way more time than CPA implementation. This performance disgrace is due to the large number of Conlon's over. For example, let's profile this implementation with in some remote. OK, here you can see that almost 70000 gun launches are happening for the implementation. This overwhelming amount of current launches are hindering the performance of the production implementation with dynamic parallelism. If we can somehow reduce the number of going alone just by keeping the same amount of parallelism, they could improve the performance of this implementation even further. So take your time and suggest me a way to improve this reduction implementation.