While deep learning has become popular, many current deep learning frameworks face a common performance problem: frequent calls to algebraic operators seriously affect the execution efficiency of the model. Common deep learning frameworks (e.g., CNTK, TensorFlow, and Caffe2) abstract a deep learning model as a directed acyclic data flow graph (DAG) composed of some basic operators, and then the lower-level computation engine sequentially schedules and executes the kernel functions corresponding to these nodes in a certain topological order to complete execution of a model. In order to support computation on different hardware, an Operator often corresponds to multiple kernel function implementations, for example, kernel functions on GPUs are a combination of operations provided by CUDA or some GPU libraries (e.g., cuDNN, cuBLAS, etc.).

In order to provide better flexibility, Operators in most deep learning frameworks are defined at the granularity of algebraic operators, such as vector addition, subtraction, multiplication, division and matrix multiplication, etc. Typical computational frameworks have hundreds or even thousands of Operators. because of the low abstraction granularity of these operators, the data flow graph of a real training model often includes thousands of nodes, and the execution of these nodes on the GPU becomes thousands of kernel executions on the GPU. While these lower-level kernel functions provide flexibility, their frequent invocations become a significant factor affecting the performance of many deep learning frameworks. The performance overhead is mainly in the scheduling overhead of the data flow graph, the startup overhead of GPU kernel functions, and the data transfer overhead between kernel functions.

A straightforward approach to solving these performance problems is Kernel Fusion, which is the fusion of the kernel functions corresponding to the nodes in a computational graph into a single function, so that the entire data flow graph can be completed with a single function call, thus reducing the overhead of platform scheduling and kernel startup. In addition, by designing the placement of input and output data for different kernel functions (e.g., using shared memory or registers on the GPU), the data transfer efficiency can be greatly improved, thus enhancing the overall computational performance.

To demonstrate the benefits that kernel fusion can bring, we compared the model inference time of an 80-step single-sample LSTM network on TensorFlow with the computation time when we manually fused all the computations and optimized them in the same kernel function. In a result, the fused kernel function can be about 40 times faster than the graph-based computation on TensorFlow on the same GPU.