November 28th Lecture Summary

2023-12675 박지호

In today's lecture, we studied more about the accelerators used for neural network models. First, we reviewed the architecture of large language models we learnt yesterday. We first looked at the attention layer and its query-key-value structure, and how each vectors are obtained from a weight matrix. It was also mentioned that these vectors can be cached and reused to reduce the number of multiplications required. We also looked at the feed-forward network where the output of attention layer is used to create the output. The combination of attention and feed-forward network layers form a transformer layer. In the model, multiple transformer layers are placed so that the next layer uses the previous layer's output as input.

For each token and for each layer, $12H^2 + 2LH$ multiplications are required. During the prefill phase, all computations can be performed in parallel since we have all the input tokens. Therefore, we only need to load the weight matrix once, and compute all the necessary vectors at once. Similarly, we only need to load the key and value vectors once from the cache as well. However, for the generation phase, we can only generate one token at a time. Therefore, such optimization is not possible.

Then, we used the roofline model to evaluate the computation speed in a real world scenario. We also discussed how to improve the performance in that scenario. First, using a DRAM with higher bandwidth would help, but that is very difficult. Using an hardware with higher TOPs would also help, but not when the bottleneck is memory access speed. Using smaller model will definitely improve performance, but that is more of a compromise than an optimization. Using lower precision for parameters is the most realistic way to improve performance, as that will significantly increase operations per second. Several other optimizations were discussed as well, but not in too much detail.

As another optimization technique, batching was introduced. Batching refers to a technique where multiple users' input are processed together. This can help by allowing data reuse for model parameters and thereby increasing the operations per byte. Once again, we looked at a real world scenario to see how much this improves performance. We found out that by using a 1K batch, it was possible to get a operations per byte higher than the threshold operations per byte. From here, since compute capability is now the bottleneck, we can further improve performance by using the 4-bit multiplication. In the end, we were able to achieve the speed almost 1000 times faster than the naive approach.

However, batching is less effective for longer input sequence, since KV needs to be generated separately for each input and cannot be reused. Also, we can't batch as many inputs for longer sequence since the main memory is limited. Although the memory limitation can somewhat be handled by using multiple GPUs, this architecture still struggles to deal with longer sequences.

Therefore, in order to handle longer sequences, we need a hardware specifically designed to process the necessary computations. First of all, the tensor core can perform small matrix multiplication and addition in one clock cycle. This helps since we usually perform a large matrix multiplication by cumulating the sum of small matrix multiplications. Each unit in the tensor core simply performs a multiplication and addition. One instance, consist of 16 units, performs the multiplication for each cell in parallel to compute the product of one column and row vectors in one clock cycle. Then, 4 instances can perform this in parallel as well, to compute the final matrix in one cycle.

When using the tensor core, we can further optimize the performance if the target matrix have a specific pattern for zero values. We can do this by ignoring the zero values to compress the matrix. This is called sparse tensor core. In order to achieve sparsity in the target matrix, we can fine tune the weight during the training process. We do this by removing weights with small magnitudes. Since this results in a quality loss, we train the model again after this. This step is repeated until we have sparse enough weight matrix.

This process is called pruning. We can also permutate the order of columns before pruning to keep more data. Also, using lower precision obviously also help improve the performance of tensor core.