QMoE: Practical Sub-1-Bit Compression of Trillion-Parameter Models

Main Idea: this paper presents a technique to quantize/compress large MoE models. More specifically, it compresses Switch Transformer (1.6T parameter model, 3.2 TBs of memory needed) 20x, to only 160 GB (0.8 bits per parameter compared to 16 without any compression). This quantization technique at the Switch scale can be done in less than a day on a single GPU and results in a minor accuracy loss.

MoE: faster inference with the tradeoff of higher memory cost.

Usually, post-training compression techniques only reduce parameters to around 3 or 4 bits. This would not be enough to make MoE practical – giving inspiration to QMoE.

MoE Quantization

- It might suffice only quantizing MoE layers (not FFs).
- Large dense models are more resistant to quantization, so MoE models can be a good target for it (due to increase in scale seeming to relate to better teachers).
- MoE training might be highly resistant to noise.

Main Challenges:

- Memory

    - The quantization process requires data. In the case of MoEs, the data needed is much larger than with dense models, due to the potential large number of experts. This means that it is even more important to have data that represents different parts of the distribution, so all experts are represented.
- GPU utilization
    - Large-scale quantization had previously been applied to dense models, which consists of applying it to single massive individual layers, which is fast and efficient on GPUs. This can be challenging for MoEs as instead of single massive layers there can potentially be many experts.

<u>QMoE Method</u>

For dense part of the model:

- Fetch one sample X, containing a few hundreds of tokens, from CPU to GPU.
- Pass it through the corresponding dense layers to obtain the result Y.
- Calculate and store expert assignments for tokens in Y.
- Send Y back to CPU and overwrite X in B (large buffer).

For sparse part of the model (expert FFs):

- Fetch all individual tokens in B that have been assigned to expert E, denoted by Xe, from CPU to GPU.
- Use these tokens to produce compressed expert E’ (for example, with GPTQ).

- Run Xe through E' to get Ye'.
- Send Ye' back to CPU and overwrite Xe in B.

In sum:

- The dense part consists of passing a set of samples X from CPU to GPU, performing a forward pass through only the dense layers, calculating the expert assignments of each input in X to know which experts were used and then storing the outputs of the forward pass and expert assignments in Y, passing it back to the CPU (in QMoE, the dense parts of the model are left uncompressed).
- The sparse part consists of performing a loop through each expert. For each expert, from buffer B, all the tokens in X assigned to that expert are taken, forming Xe. A quantized version of the expert is also formed (through a technique like GPTQ), resulting in E'. Then, Xe is passed through E', with the outputs forming Ye'. Finally, the results Ye' are stored in the buffer B, replacing the input tokens Xe.

The method described provides the main compression gains from QMoE but is not sufficient to achieve the goal of 1 bit per parameter established. To achieve this, the authors adopt GPTQ optimizations for the MoE case, GPU decoding optimizations, and more.

<u>Results</u>

- MoEs are shown to be highly robust to quantization as vanilla rounding with ternary precision does not lead to a model collapse.

- Using data-dependent quantization in MoE (method explained) allows 2-bit and ternary quantization with minimal accuracy loss.

My takeaways:

- Due to my lack of expertise in quantization methods like GPTQ, I did not find it relevant to go into this topic more in-depth.