Computation for Foundation Models

Tensor Paralellism

Grant Glaess,

November 24, 2025

What is a foundation model

A foundational model is a machine/ deep learning model that can be applied to a wide range of use cases. Examples include GPT, stable diffusion, and DALL-E. The most common architecture used currently is the transformer architecture.

Simplified Foundation Model

To begin with, we will only look at what happens to the input of a foundation model, and not the process of it backtracking and learning. This version of a "model" has a lot stripped away from it for the purpose of explaining, but the methods shown here not only can be used for a full model, but are actively being used in existing models. For parallelism analysis, it is common to treat each column of X independently.

$$y = f_n(f_{n-1}(...f_1(X)))$$

$$f_i = \sigma(W_i X + b_i)$$

Our input X is a 2d matrix, the W's are linear transformations so they can be represented as matrices, and σ is a nonlinear function, and y is the output.

Tensor Parallelism

Tensor parallelism is a way of computing our large linear operations from the previous slide across multiple processors, or gpus, using linear algebra tricks to split up the computations.

Tensor Parallelism (1D)

For our 1D case, X being a matrix doesn't actually matter and we can simplify how we think about it to just considering the processing of each column of X, which we will label x. So for 1D, our model simplifies to $f_i = \sigma(W_i x + b_i)$.

$$WX = W \begin{bmatrix} x_1 & x_2 & \cdots & x_r \end{bmatrix} = \begin{bmatrix} Wx_1 & Wx_2 & \cdots & Wx_r \end{bmatrix}$$

In other words, we will think of x as being a single column vector for our 1D tensor parallelism cases.

Tensor Parallelism(Row)

We can split the weight matrix W_i into groups of rows depending on how many processors we have. Each entry of the kth result of $W_i x$ is equal to the kth row of W_i times x, so we just have each processor compute $C_j x$. Since each processor is fully responsible for specific entries of the result, we can also give the corresponding entries of b_i, which we will label b_{L_j} to each processor and apply our nonlinear function. We end up with each processor computing $\sigma(C_j x + b_{L_j})$, and then gathering the various elements from each device into a single vector to be fed into the next function

$$W_i = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_m \end{bmatrix}$$

Tensor Parallelism(Column)

We can split the weight matrix W_i into groups of columns depending on the number of processors we have. Each A_j is stored on a different processor, and along with it we send the entries of x that correspond to the columns of W_i that are held within each A_j as a column vector, x_{L_j}.

$$W_i = \begin{bmatrix} A_1 & A_2 & \dots & A_3 \end{bmatrix}$$

Each processor is then computing $A_j x_{L_j}$ which we can think of as

$$A_j x_{L_j} = \begin{bmatrix} a_{11}(j) & a_{12}(j) & \dots & a_{1k}(j) \\ a_{21}(j) & a_{22}(j) & \dots & a_{2k}(j) \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}(j) & a_{m2}(j) & \dots & a_{mk}(j) \end{bmatrix} \begin{bmatrix} x_p \\ x_p + 1 \\ \vdots \\ x_p + k - 1 \end{bmatrix}$$

The result of each processor is then combined entry by entry to be create the final vector.

2D Tensor Parallelism

Suppose we have $WX + B$ and q^2 processors. We can split the matrices W and X into q^2 blocks each and get

$$W = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1q} \\ W_{21} & W_{22} & \cdots & W_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ W_{q1} & W_{q2} & \cdots & W_{qq} \end{bmatrix}, \qquad X = \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1q} \\ X_{21} & X_{22} & \cdots & X_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ X_{q1} & X_{q2} & \cdots & X_{qq} \end{bmatrix}.$$

$$Y = WX = \begin{bmatrix} \sum_{k=1}^{q} W_{1k}X_{k1} & \sum_{k=1}^{q} W_{1k}X_{k2} & \cdots & \sum_{k=1}^{q} W_{1k}X_{kq} \\ \sum_{k=1}^{q} W_{2k}X_{k1} & \sum_{k=1}^{q} W_{2k}X_{k2} & \cdots & \sum_{k=1}^{q} W_{2k}X_{kq} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{q} W_{qk}X_{k1} & \sum_{k=1}^{q} W_{qk}X_{k2} & \cdots & \sum_{k=1}^{q} W_{qk}X_{kq} \end{bmatrix} + B.$$

2D Tensor Parallelism pt 2 (SUMMA algorithm)

Since we have a grid of q^2 processors, and a matrix of q^2 blocks, each processor is given W_{ij} and X_{ij} and computes $\sum_{k=1}^{q} W_{ik}X_{kj}$. To do this, each processor computes $W_{ij}X_{ij}$ and broadcasts W_{ij} to each processor in the same row as it and broadcasts X_{ij} to processors in the same column. This gives each processor exactly the block matrices it needs to compute each sum.
Note: There are similar algorithms to compute both A^TB and AB^T, which are are both computations that can be found in a transformer model, the most common type of foundation model.

Performance Statistics

-2d tensor parallelism has a lower communication cost than 1d

Strengths and Weaknesses

2D tensor parallelism comes with the downside of requiring a square number of processors to work. Also, if the split is too aggressive and the matrix blocks become too small which leads to poor GPU utilization. On the other hand, 2D tensor parallelism comes with decreased communication volume which means increased communication efficiency. Since we are splitting in both dimension for 2D, each processor only needs a smaller block of W and X which reduces the amount of data exchanged. While 1D tensor parallelism is easier to implement, it doesn't scale as well as communication involves sending matrices that have one dimension proportional to the entire model size.

Relevancy, Impact, real world uses and active research

As AI models are getting bigger and bigger, we need ways to actually handle these massive models both on the hardware and software side, as well as ways to parallelize the models in the first place.
Tensor parallelism is used in many large foundation models, which are of course widely used today. Examples include chat bots and image generation, all the way to more professional use cases such as medical ones.
As for active research on computation for foundation models, almost all of the information talked about shows up in distributed training systems for models such as DeepSpeed and Megatron-LM. Both of which are actively researching new parallelism strategies.

Thank You

Questions?

Works Cited I

Sourangshu Ghosh. *Mathematical Foundations of Deep Learning*. arXiv:2502.0023v2 (2025). https://rxiv.org/pdf/2502.0023v2.pdf

UVaDLC. *Tensor Parallel Simple*. (2025). https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/scaling/JAX/tensor_parallel_simple.html

Mai L., et al. *Machine Learning Systems: Design & Implementation*. Chapter: *Pipeline Parallelism with Micro-Batching* (2024). https://openmlsys.github.io/html-en/chapter_distributed/Pipeline_Parallelism_with_Micro-Batching.html

Colossal-AI Documentation. *1D Tensor Parallelism*. (2024). https://colossalai.org/docs/features/1D_tensor_parallel

Xu, Qifan, et al. *An Efficient 2D Method for Training Super-Large Deep Learning Models*. arXiv:2104.05343 (2021). https://arxiv.org/pdf/2104.05343

Li, Liu, et al. *2.5-D Tensor Parallelism for Distributed Model Training*. arXiv:2110.14883 (2021). https://arxiv.org/pdf/2110.14883