MoE Routing Simulation and Performance Model

Background

The recent surge in popularity surrounding LLMs has given rise to models with hundreds of billions of parameters, making them too large to fit on individual GPUs. Modern models are run on distributed systems with different parallelization techniques, and as the growth rate of LLMs' computational needs exceeds the growth rate of GPU performance, it is expected that distributed HPC systems will remain the standard.

Due to a distributed system's many parallelization techniques, certain collective communication operations are required to keep the entire system, or world, synchronized.

All-Reduce

The most common operation is AllReduce, which is used during backpropagation to average the gradients from each GPU (which will be different since each GPU has not processed the same tokens/mini-batches)

AllReduce is used in every model that resides on a distributed setup since all neural networks require backpropagation and gradient averaging. Due to its widespread use, it is well optimized under certain topologies (ie: Ring) by NCCL.

All-to-All

In Mixture-of-Experts LLMs, All-to-All collectives are crucial as each GPU only contains a subset of experts. This leads to tokens from one GPU needing to be potentially sent to all other GPUs after the routing step, a costly communication operation that has its efficiency directly correlated to how balanced/uniform the routing is.

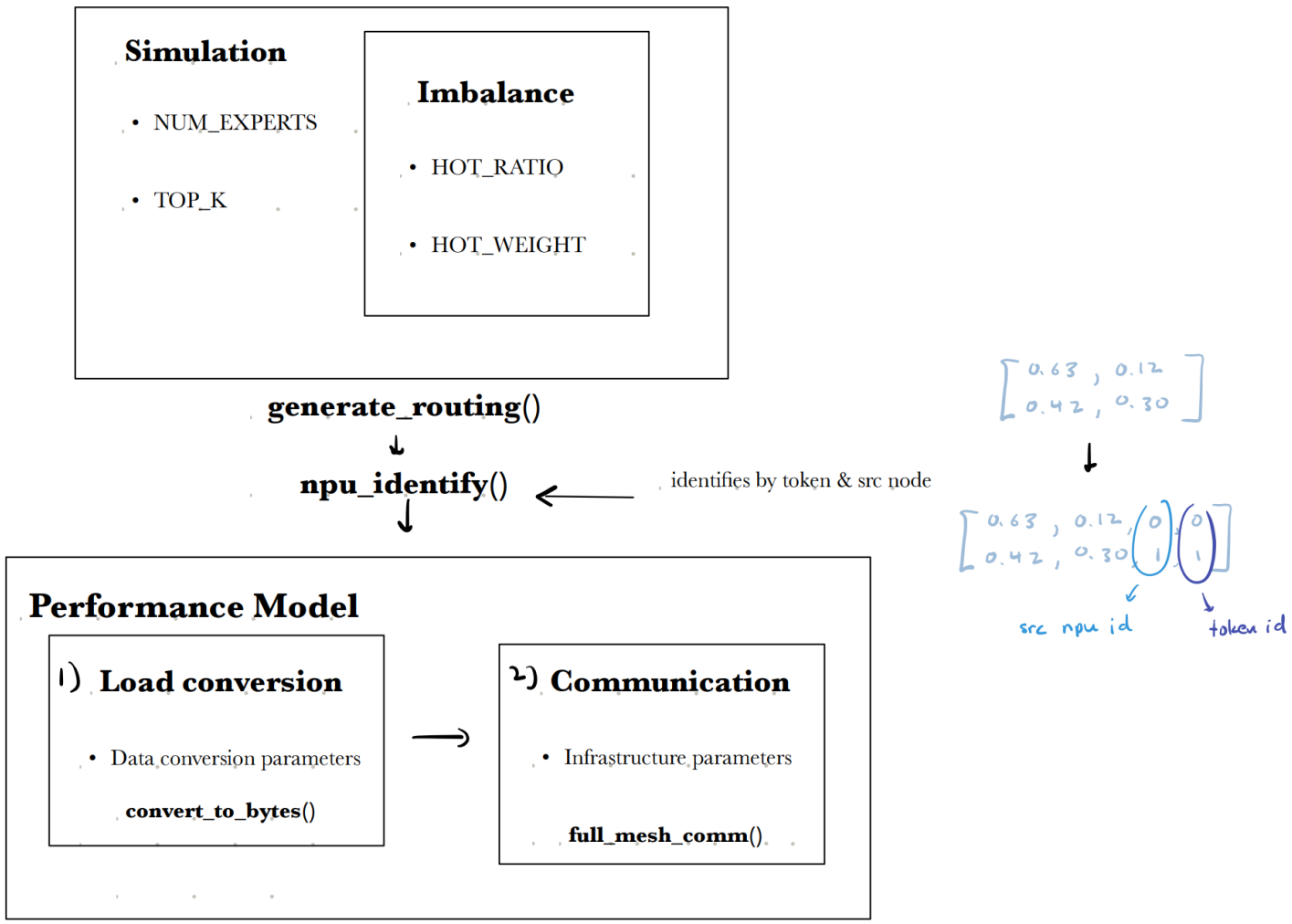
Objective

1. Router Simulation
    - Produce dummy workloads with parameters for imbalance
2. Performance Model
    - Simulate routing workloads with distributed systems parameters

Architecture Overview

1. Workflow

1. Set desired parameters and mode in params.py
2. Run simulation.py to generate routing data
3. Run perf_model.py to run communications simulation
4. See traces in comm_log.txt



2. Parameters

Simulation Parameters

NUM_EXPERTS: Number of total experts

SEQLEN: Number of tokens to process

TOP_K: Top K experts to be chosen by the router for each token

HOT_RATIO: Ratio of "hot" experts (experts picked more frequently)

HOT_WEIGHT: Weight ratio of hot experts (ie: HOT_WEIGHT = 0.8, HOT_RATIO = 0.5 means that half the experts receive 80% of the routings)

Performance Model Parameters

Data conversion

NUM_NODES: Number of GPU nodes (assuming 1 GPU per node)

WEIGHT_PRECISION: Datatype in bytes for routing weights

ROUTING_PRECISION: Datatype in bytes for destination GPU identification

ID_PRECISION: Datatype in bytes for source and sequence position identification

Infrastructure

NUM_LINKS: Number of parallel edges/links between any two nodes (>= 1 since the design is full mesh)

BASE_DELAY: Delay incurred every round

INITIAL_CPU_DELAY: Delay incurred from initial GPU and CPU communication (only happens once)

INTRA_BW: Link bandwidth between nodes in B/ms

PACKET_SIZE: Packet size in bytes

PACKET_PREP_DELAY: Packet preparation delay

ROUND_ROBIN_MAX_PACKETS: Number of packets per (src, dst) before moving on to the next pair

NIC_RATE: NIC send/receive memory constraint (cannot send/receive more than this amount per round)

Non-implemented Parameters

NUM_LAYERS: Number of layers of MoE

EMBED_DIM: Embedding dimension

INTER_BW: Bandwidth between clusters

PARALLELIZATION_MULTIPLIER: Packet preparation and communication overlap (PP)

3. Communications approach

3.1 Formula basis

With an even load, the communications time can be calculated with the following formulae:

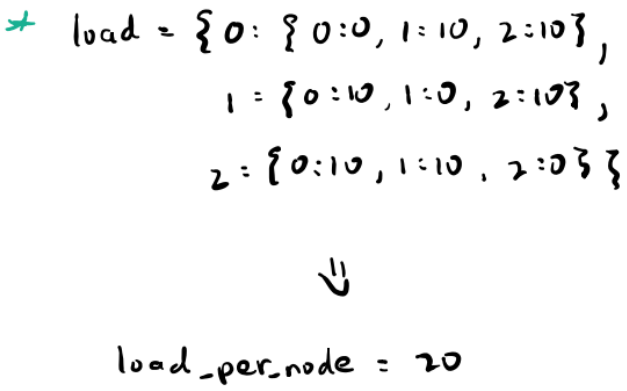
$$Round\_time = (initial\_cpu\_delay + base\_delay + \lfloor \frac{intra\_bw}{packet\_size} \rfloor) * (packet\_prep\_delay + \frac{packet\_size}{intra\_bw})$$

$$Num\_rounds = \frac{load\_per\_node * num\_nodes}{\frac{num\_links * num\_nodes * (num\_nodes-1)}{2} * intra\_bw}$$

$$Total\_time = Round\_time * Num\_rounds$$

All but one of the variables above are parameters that can be edited in parameters.py

Here is an example illustrating how load_per_node is calculated:



full_mesh_comms will give consistent results with this formula given that the load is perfectly distributed over all nodes.

3.2 Simulation approach

NIC Memory & Active link dictionaries track all of the packets sent every round, and round robin balancing is used to ensure each node's NIC memory is not completely used between two nodes with a heavy load.

Here is a round robin example, where node 0 wants to send data to 3 nodes, no NIC memory restrictions, packet_size = 2 and **round_robin_max_packets = 2**