

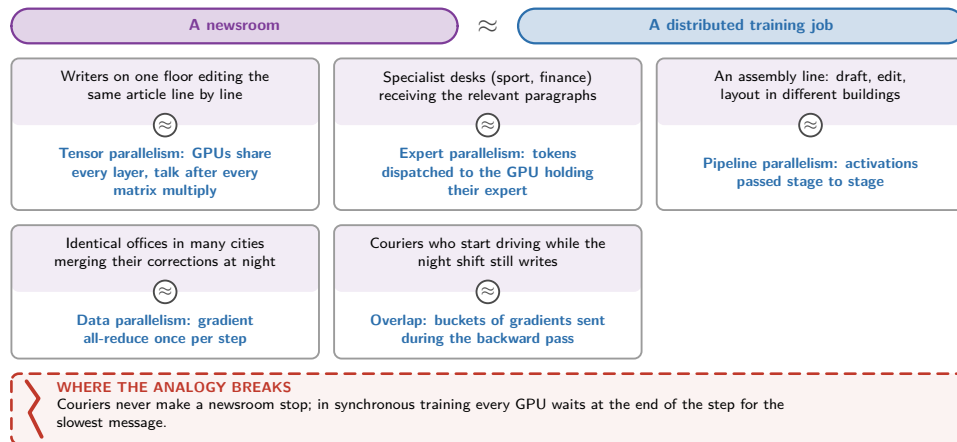


Why Networks Matter for AI

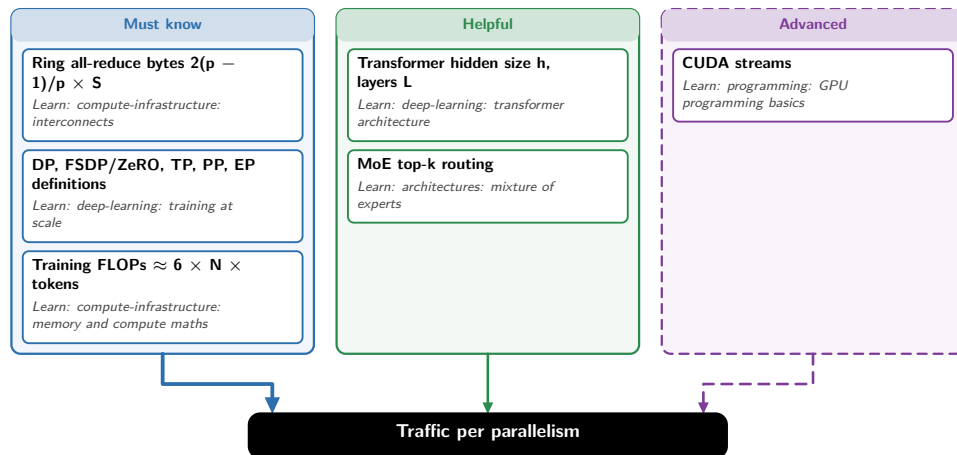
AI networking | Scale-up vs scale-out, bytes per step for data, sharded, tensor, pipeline, context and expert parallelism, communication/compute overlap and exposed communication, with a 70B tensor-parallel example worked by hand

#ScaleUp #ScaleOut #Overlap #Parallelism #Bandwidth

The idea



A newsroom spread over several buildings



Recalled, not taught here

Scale-up vs scale-out

Scale-up (NVLink, UALink, TPU ICI)

Reach: one node or rack, copper, < 2-3 m mostly
450-1,800 GB/s per GPU per direction (NVLink 4-6)
Load/store and DMA semantics: GPUs read peer memory
8 to 72 GPUs today; up to hundreds or $\approx 1,000$ announced
Carries TP, EP, context parallelism

Scale-out (InfiniBand, RoCE, Ultra Ethernet)

Reach: whole hall, optics, up to ≈ 2 km
50-100 GB/s per GPU (one 400-800 Gb/s NIC)
Message semantics: RDMA send/write/read through a NIC
Thousands to 100,000+ GPUs
Carries DP/FSDP, PP, cross-domain EP

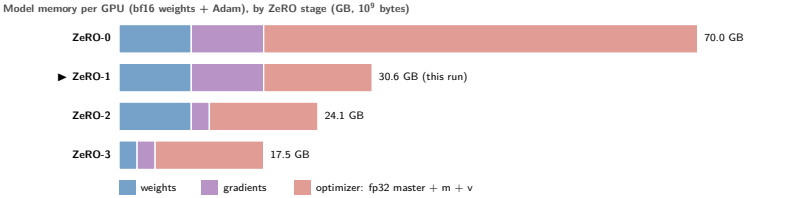
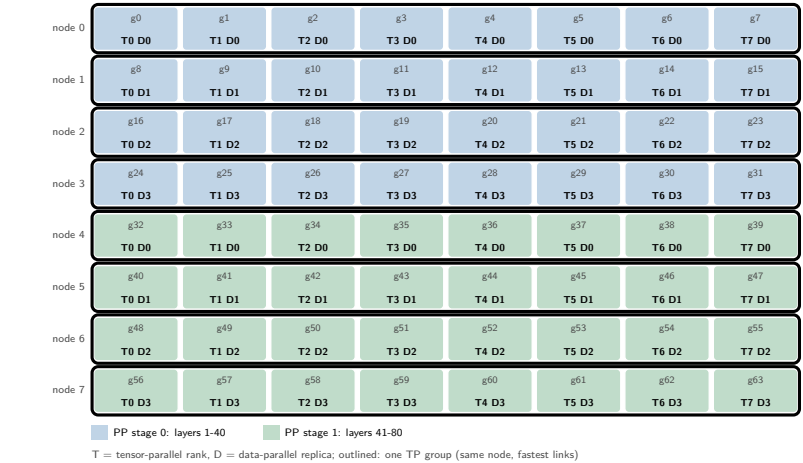
Parallelism	Collective	Bytes sent per GPU	Overlap?	Fabric
Data (DDP)	all-reduce gradients	$2(p-1)/p \times \Psi \times b$ per step	yes, during backward	scale-out
Sharded (FSDP, ZeRO-3)	all-gather W fwd and bwd, reduce-scatter grads	$3(p-1)/p \times \Psi \times b$ per step ($1.5 \times$ DDP)	yes, prefetch next layer	scale-out
Tensor (Megatron)	2 all-reduces fwd + 2 bwd per layer	$4 \times 2(t-1)/t \times m \cdot s \cdot h \cdot b$ per layer	hard: on critical path	scale-up
Pipeline	send/rcv activations	$m \cdot s \cdot h \cdot b$ per micro-batch per boundary, each way	partly (1F1B schedules)	scale-out OK
Context (ring attention)	pass K, V blocks round the ring	$\approx 2(c-1)/c \times s \cdot h_{kv} \cdot b$ per layer, forward ($c = \text{CP group size}$)	yes, with attention compute	scale-up preferred
Expert (MoE)	all-to-all dispatch + combine	$\approx 2 \times k \times \text{tokens} \times h \times b$ per MoE layer fwd ($\times 2$ bwd)	partly (two micro-batches)	scale-up preferred

Bytes per GPU per step by parallelism (ring algorithms; b = bytes per value). Recall the definitions from deep-learning; here only the traffic

Quantity	Formula	Reads
Ring all-reduce bytes per GPU	$2^{\frac{p-1}{p}} S$	$\approx 2S$ for large p
Tensor-parallel message	$S_{TP} = m s h b$	one activation tensor
MoE dispatch bytes per GPU	$\approx k T h b \frac{E - E_{loc}}{E}$	share of tokens leaving the GPU

where p = GPUs in the data-parallel group (count); t = GPUs in the tensor-parallel group (count); S = buffer size per GPU (bytes); Ψ = parameters held (or sharded) by the data-parallel group (count); b = bytes per value (2 for bf16) (bytes); m = micro-batch size (sequences); s = sequence length (tokens); h = hidden size (h_kv: key/value width) (values); k = experts chosen per token (top-k) (count); T = tokens on the GPU in this micro-batch (count); E, E_{loc} = experts in all, experts on this GPU (count)

Symbols of the traffic table



Group	size	groups	collective (per step)
TP	8	8	layer activations: 2 all-reduce per layer forward, 2 backward
PP	2	32	send / rcv activations and their gradients between stages
DP	4	16	all-reduce gradients (optimizer states sharded)

64 GPUs = DP 4 × TP 8 × PP 2 · layers per stage: 80/2 → 40 · shard per GPU before ZeRO: N / (TP × PP) = 70B / 16 = 4.38B params · pipeline bubble (1F1B / GPipe) = (pp − 1)/(m + pp − 1) = 1/9 = 11.1 %

GPUs = DP × TP × PP; per-GPU params before ZeRO = N / (TP × PP)

ZeRO-0: (b + b + K) · Ψ; ZeRO-1: (b + b + K/DP) · Ψ; ZeRO-2: (b + (b + K)/DP) · Ψ; ZeRO-3: (b + b + K) · Ψ/DP

bubble = (PP − 1) / (m + PP − 1)

where

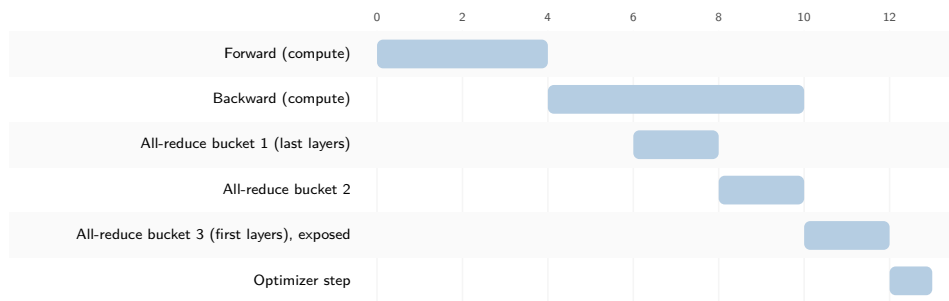
DP, TP, PP	data-, tensor-, pipeline-parallel sizes (4, 8, 2)	K	optimizer bytes per parameter: 12 (fp32 master 4 + Adam m 4 + v 4)
N	model parameters (70B)	m	micro-batches per step (8)
Ψ	parameters held by one GPU before ZeRO	GB	10 ⁹ bytes; activations not included
b	bytes per weight and per gradient (bf16 = 2 B)		

A 70B model on 64 GPUs: TP 8 inside each node (NVLink), PP 2 and DP 4 across nodes. The layout puts the chattiest group (TP) on the fastest links

⚙ Why is tensor parallelism kept inside a node?

Each transformer layer needs 2 all-reduces of a full activation tensor in the forward pass and 2 in the backward pass, and the next matrix multiply waits for them: they are on the critical path and hard to overlap. Over NVLink (≈ 450 GB/s per direction on H100) they cost a fraction of the compute; over a 400 Gb/s NIC (50 GB/s) they cost more than the compute. So TP degree $\leq$ scale-up domain size (8 per HGX node, up to 72 on NVL72).

🕒 Overlap: hiding communication behind compute



One DDP step (time units). Gradient buckets are all-reduced while the backward pass continues; the last bucket (first layers, computed last) has nothing left to hide behind: 2 units exposed

Exposed communication per step

FORMULA $E = \max(0, T_c - T_b)$

WHERE

- E = communication time no compute hides (GPU idle) (s)
- T_c = time of the step's overlappable communication, e.g. the gradient all-reduce (s)
- T_b = compute time available to overlap it (backward pass after the first bucket is ready) (s)

WORKED

$T_c = 0.56$
 $T_b = 0.3$
 $E = \max(0, 0.56 - 0.3)$
 $E = 0.26$ s idle per step

RESULT $E = 0.26$ s idle per step

READ IT 7B model, bf16 gradients over 400 Gb/s NICs, flat-ring worst case (sheet 00 example): 0.26 s of idle GPU per step. Halving T_c (800 Gb/s NICs) or doubling T_b (larger micro-batch, more gradient accumulation) brings E to 0.

MISTAKES

- Overlap needs free SMs: NCCL kernels steal a few SMs, so overlapped compute itself slows by a few %
- Gradient accumulation divides sync frequency: all-reduce once per k micro-batches
- Tensor-parallel traffic is not in T_c : it blocks the next layer

- 1 **Overlap** DDP buckets, FSDP prefetch of the next layer's weights, async pipeline send/recv
- 2 **Place by bandwidth** TP and EP inside the scale-up domain, DP and PP across it; topology-aware scheduling (sheet 03)
- 3 **Go hierarchical** reduce inside the node over NVLink, then across nodes: 8× fewer bytes per NIC
- 4 **Send fewer bytes** bf16 or fp8 communication, gradient accumulation, larger batches per sync, low-communication methods (sheet 09)
- 5 **Buy bandwidth** 800 Gb/s NICs, bigger NVLink domains, in-network reduction (SHARP)

Five levers, in the order to try them

Y

A 70B dense model ($h = 8,192$, $L = 80$ layers) trains with $TP = 8$, micro-batch $m = 1$, $s = 4,096$ tokens, bf16. Compare tensor-parallel traffic time over NVLink and over one 400 Gb/s NIC per GPU with the compute time per micro-batch (H100, 989 TFLOP/s bf16 dense, 40 % MFU).

Given: Message $S_{TP} = m \times s \times h \times b = 1 \times 4,096 \times 8,192 \times 2 \text{ B} \approx 67 \text{ MB}$; 4 all-reduces per layer (2 forward, 2 backward); ring bytes per GPU = $2(t - 1)/t \times S_{TP}$; Compute FLOPs per GPU = $6 \times N \times \text{tokens} \div t$; $N = 70 \times 10^9$ parameters

Y

Step 1: Bytes per all-reduce per GPU?

$2 \times 7/8 \times 67 \text{ MB} \approx 117 \text{ MB}$
 ring factor 1.75 for $t = 8$

T

Y

Step 2: Per micro-batch, all layers?

$117 \text{ MB} \times 4 \times 80 \approx 37.6 \text{ GB}$
 4 all-reduces per layer

T

Y

Step 3: Time over NVLink (450 GB/s each way)?

$37.6 \div 450 \approx 0.084 \text{ s}$
 H100 NVLink 4: 900 GB/s total

T

Y

Step 4: Time over one NIC (50 GB/s)?

$37.6 \div 50 \approx 0.75 \text{ s}$
 400 Gb/s $\div 8$

T

Y

Step 5: Compute per micro-batch per GPU?

$6 \times 70e9 \times 4,096 \div 8 \approx 2.15e14 \text{ FLOP}$; $\div (989e12 \times 0.4) \approx 0.54 \text{ s}$
 6N FLOPs per token, split over 8 GPUs

T

Y

Step 6: Compare?

NVLink: $0.084 / 0.54 \approx 16 \%$; NIC: $0.75 / 0.54 \approx 140 \%$
 communication-to-computation ratio

T

Over NVLink, TP costs $\approx 16 \%$ of compute (acceptable, partly hideable); across nodes it would cost $\approx 1.4 \times$ the compute: the job becomes network-bound. Keep TP inside the scale-up domain.

T

Check: Latency ($4 \times 80 \times 2 \times 7 \text{ steps} \times \text{a few } \mu\text{s} \approx 10 \text{ ms}$) is small next to the bandwidth term here.

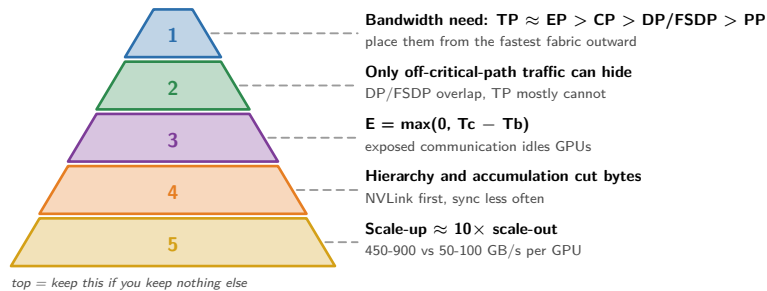
Tensor parallelism inside vs across nodes, by hand

Check and recap

Statement	True	False
1. FSDP (ZeRO-3) moves about 1.5× the bytes of plain DDP → Two all-gathers of weights plus one reduce-scatter of gradients vs one all-reduce (= reduce-scatter + all-gather).	☒	☐
2. Pipeline parallelism needs the most bandwidth of all strategies → It sends one activation tensor per micro-batch per boundary: the lightest, which is why PP spans nodes.	☐	☒
3. Gradient accumulation over 4 micro-batches divides DP traffic per sample by 4 → Gradients are synchronised once per accumulated step.	☒	☐
4. MoE all-to-all is mostly bandwidth-bound and insensitive to latency → Many small per-peer messages make it latency- and incast-sensitive, worse with many experts across nodes.	☐	☒
5. NVLink 900 GB/s on H100 means 900 GB/s in each direction → 900 GB/s is the bidirectional total: 450 GB/s each way.	☐	☒

True or false (answers and corrections shown)

Remember



Q1 Bytes per GPU per step for DDP with Ψ parameters in bf16 over p GPUs?

A $2(p-1)/p \times 2\Psi$ bytes $\approx 4\Psi$ bytes for large p .

Q2 Why does FSDP move about 1.5× DDP's bytes?

A All-gather weights in forward and again in backward plus a reduce-scatter of gradients = 3 half-all-reduces.

Q3 How many all-reduces per transformer layer in Megatron tensor parallelism?

A Two in the forward pass (attention, MLP) and two in the backward pass.

Q4 Which parallelism is the lightest on the network and usually spans nodes?

A Pipeline parallelism: one activation tensor per micro-batch per stage boundary.

Q5 Exposed communication formula?

A $E = \max(0, T_c - T_b)$: communication time minus the compute that overlaps it.

Q6 TP = 8 traffic of a 70B model ($s = 4,096$, $m = 1$) per micro-batch per GPU?

A ≈ 37.6 GB: 0.084 s over NVLink, 0.75 s over a 400 Gb/s NIC.

Q7 Two collectives of an MoE layer and their names?

A All-to-all dispatch (tokens to experts) and all-to-all combine (results back).

Q8 First lever when communication is exposed?

A Overlap it (buckets, prefetch), then place by bandwidth, then go hierarchical.

Q9 Default PyTorch DDP bucket size?

A 25 MB (bucket_cap_mb).