APPENDIX B

Performance Formulas Quick Reference

Every formula a Principal Performance Architect must be able to derive from memory · Organized by domain · With hardware constants and worked examples

---

HOW TO USE THIS APPENDIX

Every formula here appears in at least one chapter and at least one real interview or architecture review. The worked examples use real hardware specs. Variables are defined inline. Formulas marked ★ are the highest-leverage — derive them from first principles before your next interview.

1 · Roofline Model & Arithmetic Intensity ★

```
Achievable Performance = min( Peak_FLOPS, AI × Peak_BW )
```

ARITHMETIC INTENSITY

```
AI = FLOPs_executed / Bytes_transferred
   [FLOP/byte]
```

FLOPs = computation done. Bytes = DRAM traffic (not L2).

RIDGE POINT ★

```
AI_ridge = Peak_FLOPS / Peak_BW

H100 SXM:  989 TFLOPS / 3.35 TB/s = 295 FLOP/byte
MI300X:   1307 TFLOPS / 5.3  TB/s = 247 FLOP/byte
B200:     ~4500 TFLOPS / 8.0  TB/s = 563 FLOP/byte
```

If AI < ridge → memory-bound. If AI > ridge → compute-bound.

MEMORY-BOUND KERNEL TIME

```
T_mem = Bytes_transferred / Peak_BW

Example: 1 GB at 3.35 TB/s
  T_mem = 1×10⁹ / 3.35×10¹² = 0.299 ms
```

COMPUTE-BOUND KERNEL TIME

```
T_cmp = FLOPs / Peak_TFLOPS

Example: 2 TFLOP on H100 BF16
  T_cmp = 2×10¹² / 989×10¹² = 2.02 ms
```

```
# PERFORMANCE REGIME CLASSIFICATION — DECISION RULE:
IF AI < AI_ridge:  MEMORY-BANDWIDTH BOUND
    → Performance limited by HBM bandwidth
    → Fix: reduce bytes (quantization, fusion), not FLOPs
    → Achieved FLOPS = AI × Peak_BW  (not peak compute)

IF AI > AI_ridge:  COMPUTE BOUND
    → Performance limited by TFLOPS
    → Fix: increase Tensor Core utilization, use FP8/BF16
    → Achieved FLOPS ≈ Peak_FLOPS × occupancy_factor

# COMMON OPERATION ARITHMETIC INTENSITIES (H100 BF16):
Operation              AI          Regime (H100)     Achieved TFLOPS
──────────────────────────────────────────────────────────────────────
GEMV batch=1           ~1           Deep BW-bound      ~3.3
LayerNorm, RMSNorm    ~1.25        BW-bound           ~4.2
Softmax                ~5          BW-bound           ~17
GEMV batch=32          ~32         BW-bound           ~107
FlashAttn S=2048       ~250        Near ridge         ~700
GEMM M=N=K=4096       ~1,365       Compute-bound      ~800  (80% MFU)
Large GEMM FP8        ~2,730       Compute-bound      ~1,500
```

2 · GEMM Performance ★

GEMM FLOPS ★

FLOPs = 2 × M × K × N

M = output rows (batch × seq)
K = inner dimension (hidden)
N = output cols (hidden or vocab)

Factor of 2: one multiply + one accumulate per element.

GEMM ARITHMETIC INTENSITY

AI_GEMM = 2MKN / ((MK + KN + MN) × bytes)

Large M,N,K: AI ≈ 2K/(bytes) × (MN/(M+N))
Batch=1 (decode GEMV): AI ≈ 2 × (very small M)
→ always memory-bound at batch=1

```
# WORKED EXAMPLE: LLaMA-3 70B decode GEMV (single token, H100)
M=1, K=8192, N=8192, BF16 (2 bytes)

FLOPs = 2 × 1 × 8192 × 8192 = 134.2 MFLOPs
Bytes = (1×8192 + 8192×8192 + 1×8192) × 2 ≈ 134.2 MB  (weights dominate)
AI    = 134.2 MFLOPs / 134.2 MB = ~1.0 FLOP/byte  ← deeply BW-bound

T_mem = 134.2 MB / 3,350 GB/s = 0.040 ms per layer
80 layers × 0.040 ms = ~3.2 ms per decode step
Tokens/sec ≈ 1 / 0.0032 = ~313 tok/s  (single request, H100)
```

3 · Memory Sizing ★

MODEL MEMORY (INFERENCE)

```
M_model = P × bytes_per_param

BF16: bytes = 2    → 70B × 2 = 140 GB
FP8:  bytes = 1    → 70B × 1 = 70  GB
INT4: bytes = 0.5  → 70B × 0.5 = 35 GB
```

TRAINING MEMORY PER GPU

```
M_train ≈ P × (2 + 4 + 4 + 4) bytes
          = P × 14 bytes  (BF16 + FP32 Adam)

= weights(BF16) + grads(BF16)
  + Adam_m(FP32) + Adam_v(FP32)

70B × 14 = 980 GB total (before sharding)
```

Requires ZeRO-3 / FSDP across many GPUs.

KV CACHE SIZE ★

```
KV_bytes = 2 × L × H_kv × D_head × S × dtype

L    = num layers
H_kv = num KV heads (GQA-aware)
D    = head dimension (hidden/H)
S    = sequence length
dtype: BF16=2, FP8=1

LLaMA-3 70B (GQA, 8 KV heads):
  2 × 80 × 8 × 128 × 4096 × 2 = ~13 GB
  per request at 4K context
```

KV CACHE BATCH CAPACITY

```
Max_batch = (HBM - M_model) / KV_per_request

H100 80GB, LLaMA-3 70B BF16:
  Available = 80 - 140 = NEGATIVE → use TP

H100 × 2 (TP=2): Available = 160 - 140 = 20 GB
Max_batch = 20 GB / 13 GB ≈ 1.5 → need FP8 or longer context chunking
```

4 · Training Performance ★

TRAINING FLOPS PER TOKEN ★

$C_{token} \approx 6 \times P$ (forward + backward)

Forward: 2P FLOPs/token
Backward: 4P FLOPs/token
Total: 6P FLOPs/token

70B model: $6 \times 70 \times 10^9 = 420$ GFLOPs/token

Approximation; exact count depends on architecture (attn, FFN ratios).

MFU — MODEL FLOP UTILIZATION ★

$MFU = (tok/s \times C_{token}) / (N_{GPU} \times F_{peak})$

tok/s = measured throughput
N_GPU = total GPU count
F_peak = peak TFLOPS per GPU

H100, 8 GPU, 70B, 1000 tok/s:
$= (1000 \times 420 \times 10^9) / (8 \times 989 \times 10^{12})$
$= 420 \times 10^{12} / 7912 \times 10^{12}$
$= 5.3\%$ (too low → investigate)

MFU Range	Assessment	Typical Root Cause
>55%	**Excellent**	Well-configured large GEMM training
40–55%	**Good**	Expected for most production training runs
25–40%	**Suboptimal**	Communication overhead, small batch, PP bubbles
<25%	**Poor**	NCCL issues, memory OOM forcing small batches, straggler

```
# TRAINING FLOP COUNT — COMPLETE MODEL (LLaMA-3 70B, 8192 seq)
Components (per token, BF16):
  QKV projection:  2 × 3 × H² × L = 2 × 3 × 8192² × 80 = 32.2 TF
  Attention:       4 × H × S × L  = 4 × 8192 × 8192 × 80 = 21.5 TF (quadratic in S!)
  O projection:    2 × H² × L     = 2 × 8192² × 80 = 10.7 TF
  FFN (GeLU):      2 × 3.5 × H² × L = 2 × 3.5 × 8192² × 80 = 37.6 TF
  Total forward:   ≈ 102 TF/token
  × 6 (fwd+bwd):  ≈ 613 TF/token  (vs 6P=420 TF — attn adds ~46% at S=8K)

Key insight: Attention cost scales as O(S²), GEMM as O(S).
At S=128K (long context), attention dominates — FlashAttention is essential.
```

5 · Inference Latency & Throughput ★

TTFT — TIME TO FIRST TOKEN

```
TTFT ≈ S_in × T_step  (prefill)

T_step = max(T_compute, T_memory)

Prefill: compute-bound at large S
  T_compute = 2 × P × S / (N × F_peak)

LLaMA-3 70B, S=1024, 8×H100:
  = 2×70B×1024 / (8×989T) = 18.2 ms
```

TPOT — TIME PER OUTPUT TOKEN

```
TPOT ≈ 2P / (N × Peak_BW)   (BW-bound)

= bytes_of_weights / bandwidth

H100 (TP=1), 70B BF16:
  = 140 GB / 3.35 TB/s = 41.8 ms/tok
  = ~24 tok/s

H100 (TP=2), 70B BF16:
  = 70 GB / 3.35 TB/s = 20.9 ms/tok
  = ~48 tok/s
```

THROUGHPUT VS LATENCY

```
Throughput = Batch × (1/TPOT)
Latency    = TTFT + S_out × TPOT

Continuous batching sweet spot:
  Target: TPOT × Batch ≤ SLA_latency

At TPOT=20ms, SLA=2s output:
  Max output = 2000/20 = 100 tokens
  Batch can grow until TPOT hits limit
```

MFU FOR INFERENCE

```
MFU_inf = (tok/s × C_fwd) / (N × F_peak)

C_fwd ≈ 2P per token (forward only)

Good inference MFU: 40–60%
(lower than training — BW-bound decode)
```

```
# SPECULATIVE DECODING SPEEDUP
Acceptance rate α (0 to 1):
  If draft proposes γ tokens, accepted tokens k ≈ γα

Speedup = (1 + α + α² + ... + αᵞ) / (1 + γ/r)
        where r = target_speed / draft_speed

Practical rule: at α=0.8, γ=4: speedup ≈ 2.5–3×
              at α=0.6, γ=4: speedup ≈ 1.8–2×
              Speedup falls to 1× when α → 0 (poor draft alignment)

# CONTINUOUS BATCHING — BATCH CAPACITY
GPU_memory = M_model + Batch × KV_per_request + activation_overhead
KV_available = GPU_memory - M_model - overhead
Max_batch    = KV_available / KV_per_request
```

6 · KV Cache & Attention ★

```
# KV CACHE MEMORY — FULL DERIVATION
Per-token KV storage:
  Each layer:  K tensor [H_kv, D_head] + V tensor [H_kv, D_head]
               = 2 × H_kv × D_head × dtype_bytes

Full sequence (length S), all layers:
  KV_bytes = 2 × L × H_kv × D_head × S × dtype_bytes

MODEL COMPARISON (4K context, BF16):
Model          L    H_kv  D_head   KV (4K)   KV (128K)
──────────────────────────────────────────────────────
LLaMA-3  8B   32    8     128    0.5 GB     16 GB
LLaMA-3 70B   80    8     128    1.6 GB     52 GB
LLaMA-3 70B   80    8     128    1.6 GB     52 GB  (GQA: 8 not 64 heads!)
GPT-4 (est)  ~96   ~16   ~128   ~4.0 GB    ~128 GB
Mixtral 8x7B  32   8     128    0.5 GB     16 GB  (MoE: same KV as 7B)

# FLASHATTENTION vs STANDARD ATTENTION MEMORY
Standard attention:  O(N²) — stores full N×N score matrix in HBM
FlashAttention:      O(N)  — tiles across SRAM, never writes full matrix

Memory saved at S=8192 (BF16, 80 heads):
  Standard: 8192² × 80 × 2 = 10.7 GB (just score matrix!)
  Flash:    Tiled → SRAM only → ~0 HBM for scores

Speedup: 2–4× at S=2K–8K. Required for S>16K on H100.
```

7 · Parallelism Strategy ★

TENSOR PARALLELISM (TP)

```
Memory_per_GPU = M_model / TP
Latency_per_layer = T_base / TP
                  +
T_AllReduce(TP)

AllReduce cost: 2 × hidden / bandwidth
Constraint: keep TP within NVLink domain
Rule: TP ≤ 8 (single node)
```

PIPELINE PARALLELISM (PP)

```
Pipeline bubble fraction:
  f_bubble = (PP - 1) / (PP - 1 + m)
  m = micro-batches per step

At m=8, PP=4:
  f_bubble = 3/11 = 27%  (high waste!)

Target: m ≥ 4×PP for bubble <20%
```

ZERO MEMORY SAVINGS

```
ZeRO-1: shard optimizer states
  Saving = (K-1)/K × M_optim (K=world size)
ZeRO-2: + shard gradients
  Saving += (K-1)/K × M_grads
ZeRO-3: + shard parameters
  Saving = (all states)/K

For K=64 GPUs (≈100% saving per GPU):
  70B model: 980 GB / 64 = 15.3 GB/GPU
```

4D PARALLELISM DECISION

```
1. Model doesn't fit 1 GPU?
      → TP first (fast NVLink)
2. Still doesn't fit after TP=8?
      → PP across nodes
3. Need more throughput?
      → DP (replicate the TP+PP unit)
4. Very long sequences?
      → SP (sequence parallelism)

Target: TP×PP×DP×SP = total GPUs
```

Strategy	Shard Dim	Comms Required	Latency Impact	Use When
Tensor Parallel	Model weights	AllReduce / AllGather per layer	**High (per-layer)**	Model > single GPU HBM
Pipeline Parallel	Model layers	P2P send/recv (small)	**Low (bubble only)**	Very large models, many nodes
Data Parallel	Batch	AllReduce gradients	**Low (<10% typical)**	Always combined with TP/PP
Expert Parallel	MoE experts	AlltoAll (expert routing)	**Medium**	MoE models only

8 · Networking & Collective Communication ★

```
# ALLREDUCE TIME (Ring-AllReduce algorithm)
T_AllReduce = 2 × (N-1)/N × msg_size / bandwidth
            ≈ 2 × msg_size / bandwidth  (for large N)

Example: Gradient AllReduce, 70B model, 1024 GPUs, InfiniBand 400G
  Gradient size: 70B × 2 bytes = 140 GB
  Bandwidth per link: 400 Gbps = 50 GB/s
  T_AllReduce = 2 × 140 GB / 50 GB/s = 5.6 seconds  ← terrible!

  With gradient compression (10:1): 140 GB → 14 GB
  T_AllReduce = 2 × 14 GB / 50 GB/s = 0.56 seconds

# COMPUTE-COMMUNICATION OVERLAP REQUIREMENT
For efficient training: T_compute > T_AllReduce
  If T_AllReduce > T_compute → communication-bottlenecked

Overlap ratio = T_compute / T_AllReduce
  >1.5 → healthy overlap possible
  <1.0 → communication is the bottleneck, scale-up hit ceiling

# INCAST AND CONGESTION
InfiniBand Fat-Tree: bisection bandwidth = leaf_count × link_BW / 2
At 1024 nodes, 400G uplinks, 2:1 oversubscription:
  Bisection BW = 1024 × 50 GB/s / 2 = 25.6 TB/s aggregate
  Per-node share = 25 GB/s (less than full 50 GB/s in the worst case)
```

9 · TCO & Capacity Planning

```
# COST PER TOKEN (inference)
Cost_per_token = GPU_cost_per_hour / tokens_per_hour

tokens_per_hour = tok/s × 3600

Example: H100 at $3.50/hr, 24 tok/s (70B BF16, single GPU):
  tokens/hr = 24 × 3600 = 86,400
  Cost = $3.50 / 86,400 = $0.0000405/token
       = $40.50 per million tokens

With batching (batch=32): 24 × 32 = 768 tok/s
  Cost = $3.50 / (768 × 3600) = $1.27 per million tokens

# GPU CLUSTER POWER & COOLING
Total_power = N_GPU × TDP + N_CPU × CPU_TDP + networking + cooling_overhead
PUE (Power Usage Effectiveness) = Total_facility_power / IT_equipment_power
Cooling_overhead = Total_power × (PUE - 1)

Target PUE: <1.2 (liquid cooling), 1.3–1.5 (air cooling)
At 10,000 H100 × 700W each = 7 MW compute power
With PUE=1.2: Total facility = 8.4 MW

# CHECKPOINT OPTIMAL INTERVAL
T_interval_optimal = sqrt(2 × T_ckpt / P_fail_per_hour)

Where P_fail = expected failures per GPU-hour × N_GPUs
T_ckpt = checkpoint write time (minutes)

Example: 10K GPU cluster, P_fail=0.07/hr, T_ckpt=2min:
  T_interval = sqrt(2 × 2/60 / 0.07) ≈ 58 minutes
```

10 · Hardware Quick Reference

GPU	BF16 TFLOPS	FP8 TFLOPS	HBM BW	HBM Size	NVLink BW	Ridge (BF16)
A100 SXM	312	—	2.0 TB/s	80 GB	600 GB/s	156 F/B
H100 SXM	989	1,979	3.35 TB/s	80 GB	900 GB/s	**295 F/B**
H200 SXM	989	1,979	4.8 TB/s	141 GB	900 GB/s	206 F/B
B200 SXM	~4,500	~9,000	8.0 TB/s	192 GB	1,800 GB/s	**563 F/B**

GPU	BF16 TFLOPS	FP8 TFLOPS	HBM BW	HBM Size	NVLink BW	Ridge (BF16)
MI300X	1,307	2,614	5.3 TB/s	192 GB	896 GB/s	**247 F/B**
MI325X	1,307	2,614	6.0 TB/s	288 GB	896 GB/s	218 F/B

Interconnect	Max BW	Topology	Latency	Use Case
NVLink 4 (H100)	900 GB/s	All-to-all (NVSwitch)	~1 µs	Intra-node GPU comm
NVLink 5 (B200)	1,800 GB/s	All-to-all	<1 µs	Intra-node
NVLink-C2C	900 GB/s	Point-to-point coherent	~3 µs	Grace CPU ↔ Blackwell GPU
InfiniBand NDR	400 Gbps (50 GB/s)	Fat-tree	~1 µs	Inter-node HPC/AI
RoCEv2 (Spectrum-X)	400 Gbps	Ethernet fabric	~2-5 µs	Inter-node (cheaper)
PCIe 5.0 ×16	128 GB/s	Point-to-point	~10 µs	CPU ↔ GPU (PCIe form)

FORMULA DERIVATION CHECKLIST — INTERVIEW READY

Before any performance interview, be able to derive from scratch without notes: (1) Roofline model and ridge point for H100. (2) MFU formula and worked example. (3) KV cache size for any LLM. (4) TPOT for a given model and hardware. (5) AllReduce time for a given message size and bandwidth. These five cover 80% of AI infrastructure system design questions.