PART IV · CHAPTER 10

Training Systems at Scale

From Data Pipeline to Fault Recovery — Engineering Reliable, High-Throughput Training for Hundred-Billion-Parameter Models

Book: AI/ML Infrastructure from Silicon to Scale | **Author:** Venkat Vinjam | **Cross-ref:** *HBM memory (Ch 4) · GPU kernels (Ch 7) · Quantization (Ch 8) · Compiler stack (Ch 9) · Networking/NCCL (Ch 14) · Cluster mgmt (Ch 16) · Telemetry (Ch 17)*

> *"Anyone can run a training job on eight GPUs. The discipline is making 8,192 GPUs behave like one — reliably, efficiently, and continuously — for months at a time."*
>
> — AI Infrastructure Engineering Maxim

Training large language models at scale is one of the most demanding distributed systems problems in existence. A single 70B parameter pretraining run consumes tens of thousands of GPU-days, generates petabytes of gradient traffic, and must operate continuously for weeks without interruption despite hardware failures, software bugs, and numerical instabilities that emerge only at scale. The difference between a cluster running at 35% MFU and 65% MFU is the difference between $18M and $10M for the same training run.

This chapter is the training systems counterpart to Chapter 14 (networking) and Chapter 16 (cluster management). Where those chapters cover *how NCCL moves data* and *how SLURM places jobs*, this chapter covers *how to architect and optimize the training system itself*. We use a running example throughout: configuring and optimizing a 70B transformer pretraining run on 512 NVIDIA H100 SXM5 GPUs connected via NVLink (intra-node) and InfiniBand NDR 400G (inter-node). Every number is derived from real hardware specifications.

FOUNDATIONAL

- Name the six training stack layers and what bottleneck each produces
- Explain the four parallelism dimensions (TP, PP, DP, CP) in plain terms
- Understand why ZeRO Stage 3 is a mathematical necessity, not an optimization
- Know why TP must never cross InfiniBand

INTERMEDIATE

- Calculate per-GPU memory at every ZeRO stage for any model size
- Compute pipeline bubble fraction and determine optimal M
- Derive MFU from tokens/sec and hardware peak TFLOPS
- Configure FSDP overlap with correct backward_prefetch flags

PRINCIPAL LEVEL

- Design complete 4D parallelism configuration for any model/cluster
- Build a fault tolerance architecture with <5% checkpointing overhead
- Construct an MFU waterfall from 100% to achieved and identify each loss source
- Size and configure RLHF infrastructure for PPO with 70B+ policy models

10.1 The Training Stack — Six Layers

A production training system is a six-layer stack in which every layer can become the bottleneck. The system runs at the speed of its slowest component. The first skill of a Principal Performance Architect is identifying *exactly* where the bottleneck lives before touching a single configuration parameter.

```
TRAINING SYSTEM STACK — SIX LAYERS:

Layer 6: GPU HARDWARE
  Tensor Core execution, HBM bandwidth, NVLink/NVSwitch fabric
  Signals: MFU <40%, high SM idle%, HBM BW saturation
  Tools: Nsight Compute, DCGM, nvidia-smi

Layer 5: COMMUNICATION LIBRARY (NCCL / RCCL)
  AllReduce, AllGather, ReduceScatter, All-to-All collectives
  Signals: NCCL ops appear AFTER compute in Nsight trace (sequential)
  Tools: nsys profile --trace=nccl, NCCL_DEBUG=INFO

Layer 4: TRAINING FRAMEWORK (Megatron-LM / FSDP / DeepSpeed)
  4D parallelism, ZeRO, gradient accumulation
  Signals: Python overhead, large PP bubbles, no overlap
  Tools: torch.profiler, NVTX regions

Layer 3: JOB SCHEDULER (SLURM / Kubernetes)
  GPU placement, topology-aware scheduling, auto-restart
  Signals: >10 min restart time, TP traffic crossing IB
  Tools: squeue, nvidia-smi topo -m

Layer 2: DATA PIPELINE (DataLoader, tokenizer, prefetch)
  Must deliver tokens at exactly the GPU consumption rate
  Signals: GPU idle between batches, CPU at 100%
  Tools: vmstat, iostat, torch.profiler DataLoader trace

Layer 1: DATA STORAGE (Lustre/GPFS/WekaFS, object store)
  Serves petabytes of tokenized shards to many readers
  Signals: IOPS saturation, metadata thrashing, open latency spikes
  Tools: lctl stats (Lustre), wekawatch, iostat

RULE: Profile all six before optimizing any one.
Most common mistake: kernel-tuning when the data pipeline is the bottleneck.
GPU shows 95% SM utilization but sits idle 30% of the time waiting for batches.
```

10.1.3 Data Pipeline Bandwidth Requirement

```
DATA PIPELINE BANDWIDTH REQUIREMENT:

BW_total = batch_size × seq_len × token_bytes × steps_per_sec

70B on 512× H100: batch=2048, seq=4096, token_bytes=2, steps_per_sec=0.5
BW_total = 2048 × 4096 × 2 × 0.5 = 8 MB/s ← trivially small!

REAL BOTTLENECK IS METADATA IOPS:
  At 1M tokens/shard: 4 file reads/sec
  With 10,000 DataLoader workers: ~40,000 metadata ops/sec
  Lustre MDT saturation: ~50,000 metadata ops/sec → metadata is the limit!

DESIGN RULES:
  1. Pre-tokenize to large binary shards (>100M tokens/file)
  2. Memory-mapped files (np.memmap) for O(1) random access
  3. Shuffle at shard level, not token level
  4. Async prefetch: num_workers ≥ 4, pin_memory=True
  5. Sequence packing for variable-length documents (+20% throughput)
```

10.2 — 4D Parallelism Deep Dive

Modern large-model training uses four orthogonal parallelism strategies simultaneously. Each operates at a different granularity and requires a different interconnect. Wrong choices can cut MFU in half — and the error is often invisible in logs until you measure step time vs. theoretical minimum.

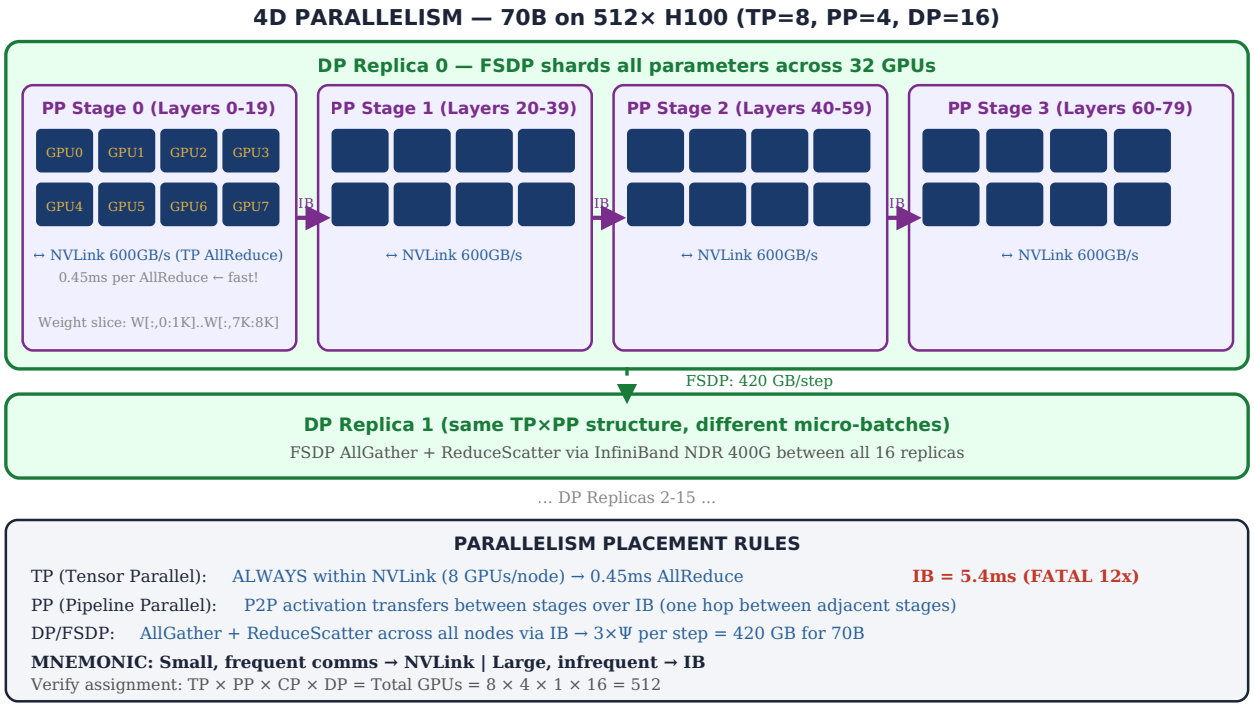


Figure 10.1 *— 4D parallelism decomposition for 70B on 512× H100. TP=8 splits each weight matrix column/row-wise within one NVLink node. PP=4 assigns 20 layers per stage, with P2P activation transfers over IB between stages. FSDP (ZeRO-3) shards all parameters across all 16 DP replicas over IB. The golden rule: TP AllReduce (268 MB × 160 calls/step) must stay on NVLink — crossing IB multiplies latency 12×.*

10.2.1 Tensor Parallelism — Column/Row Split

```
TENSOR PARALLELISM (Megatron column/row split):
  Column-parallel (QKV proj, FFN first linear):
    GPU i holds W[:, i*d/TP : (i+1)*d/TP]
    Y_i = X * W_i  (no communication in forward)
  Row-parallel (output proj, FFN second linear):
    GPU i holds W[i*d/TP : (i+1)*d/TP, :]
    Z = AllReduce(Z_0 + Z_1 + ... + Z_TP)  ← ONE AllReduce per layer

AllReduce volume (70B, TP=8, micro-batch=4, seq=4096):
  4 * 4096 * 8192 * 2 bytes = 268 MB per AllReduce
  NVLink 600 GB/s: 268/600000 = 0.45 ms  ✅
  IB NDR 50 GB/s:  268/50000  = 5.4 ms   ❌ FATAL

  80 layers * 2 AllReduces/layer:
    NVLink: 80 * 2 * 0.45 ms =  72 ms/step  ✅
    IB:     80 * 2 * 5.4 ms  = 864 ms/step  ❌ DESTROYS MFU
```

10.2.2 Sequence Parallelism

```
SEQUENCE PARALLELISM (Megatron-LM --sequence-parallel):
  LayerNorm + Dropout sharded along sequence dim → each GPU holds [seq/TP, d_model]
  Extra comms: AllGather before column-parallel, ReduceScatter after row-parallel

  Savings (70B, 80 layers, TP=8, micro-batch=4):
    Without SP: ~10.7 GB replicated activations per GPU (8 copies of same data)
    With SP:    ~1.34 GB per GPU → 8 GB freed per GPU

  RULE: Always enable when TP > 1. One flag: --sequence-parallel
```

10.2.3 Pipeline Parallelism and the Bubble Formula

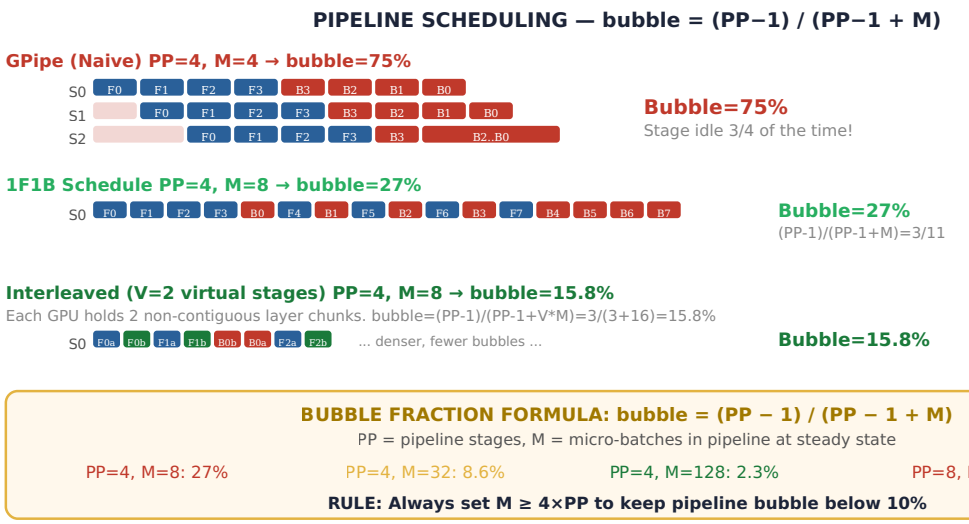


Figure 10.2 *— Pipeline scheduling comparison. Naive (GPipe): stages wait for all forward passes before beginning backward — 75% bubble at PP=4, M=4. 1F1B: alternates one forward and one backward per stage — 27% bubble at M=8. Interleaved (V=2 virtual stages): each GPU holds two non-contiguous layer chunks, denser scheduling — 15.8% bubble at same M=8. Formula: bubble = (PP−1)/(PP−1+M). Always set M≥4×PP.*

10.2.4 Data Parallelism and FSDP

```
EFFECTIVE BATCH SIZE:
  effective_batch = micro_batch × gradient_accum_steps × DP_degree

  70B on 512×H100 (DP=16, TP=8, PP=4): micro_batch=2, grad_accum=8
    effective_batch = 2 × 8 × 16 = 256 seq × 4096 = 1M tokens/update

  Pipeline micro-batch count: M = grad_accum × DP / PP = 8×16/4 = 32
  Bubble = (4-1)/(4-1+32) = 3/35 = 8.6% ✅

FSDP CONFIGURATION (PyTorch ZeRO Stage 3):
  model = FSDP(model,
    sharding_strategy=ShardingStrategy.FULL_SHARD,
    backward_prefetch=BackwardPrefetch.BACKWARD_PRE,   # overlap CRITICAL
    forward_prefetch=True,
    use_orig_params=True)     # required for torch.compile composability
```

10.2.5 Context Parallelism — Long-Sequence Training

```
CONTEXT PARALLELISM (Ring Attention) — use when seq > 32K:

PROBLEM: self-attention memory = O(seq^2)
  seq=128K, 80 layers: QK^T = 128K^2 * 2 bytes per layer = 32 GB * 80 = 2.56 TB ❌

RING ATTENTION: shard sequence across CP GPUs, rotate K,V around ring
  Memory: (seq/CP)^2 → 64x reduction for CP=8
  Comm per layer = 2 * seq * d_model * 2 bytes
    = 2 * 128K * 8192 * 2 = 4.3 GB (manageable over NVLink)

  When to use: CP=1 (seq≤32K), CP=2 (32K-64K), CP=4 (64K-128K), CP=8 (128K+)
```

10.2.6 Expert Parallelism — MoE Routing

```
EXPERT PARALLELISM (EP) — All-to-All routing:
  Volume = tokens/GPU * (EP-1)/EP * K * d_model * 2 bytes
  Mixtral-8x7B EP=8: ~56 MB per All-to-All (dispatch + combine = 2 calls/layer)
  Unlike TP: All-to-All is bandwidth-bound → IB is acceptable for EP
  Load balancing: auxiliary loss + expert capacity factor monitoring
```

10.2.7 4D Configuration Decision Tree

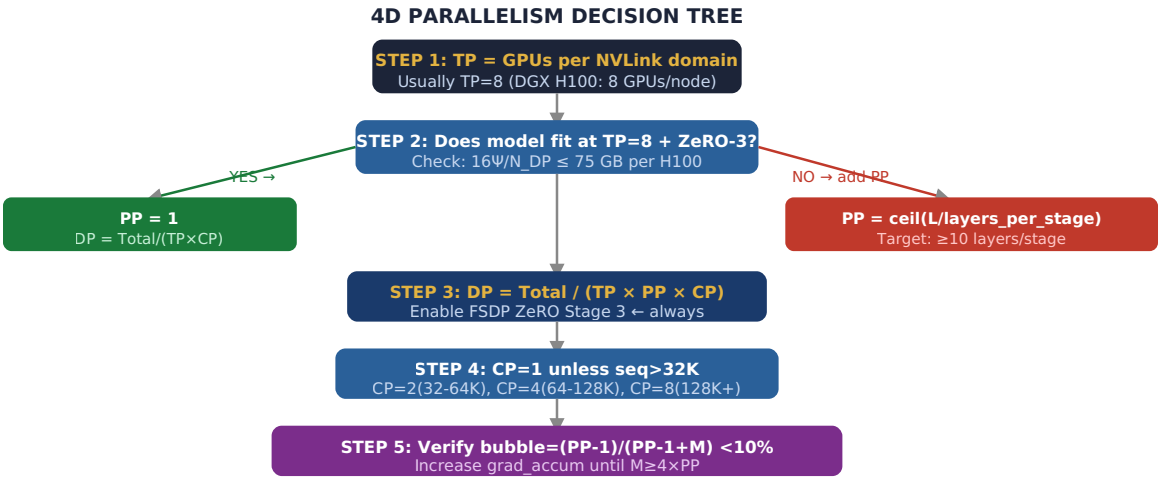


Figure 10.3 *— 4D parallelism decision tree. TP = NVLink domain (usually 8). PP only if model doesn't fit. DP gets remaining GPUs. CP only for seq>32K. Always verify bubble fraction.*

10.3 — ZeRO and FSDP: The Memory Math

10.3.1 Complete Memory Budget

```
TRAINING MEMORY COMPONENTS (per GPU, full Adam, 70B, NO sharding):
  Component             Formula            70B Amount
  ──────────────────────────────────────
  Weights BF16:         P × 2 bytes          140 GB
  Gradients BF16:       P × 2 bytes          140 GB
  FP32 master weights:  P × 4 bytes          280 GB ←
  FP32 momentum m:      P × 4 bytes          280 GB ← optimizer = 840 GB
  FP32 variance v:      P × 4 bytes          280 GB ←
  ──────────────────────────────────────
  TOTAL STATES:         P × 16 bytes        1,120 GB   ← 16 bytes/param rule!
  Activations (selec.): ~11-22 GB            22 GB (batch=4)

  16 bytes/param is the number to remember: 70B × 16 = 1,120 GB ← 14× H100 capacity
```

10.3.2 ZeRO Stage Breakdown

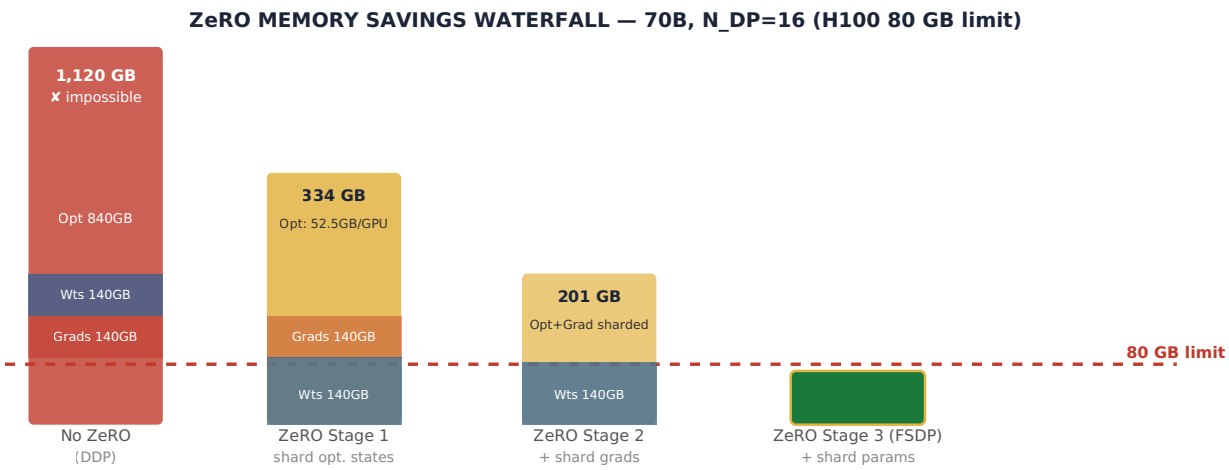


Figure 10.4 *— ZeRO memory savings waterfall for 70B, N_DP=16. Without sharding: 1,120 GB/GPU. ZeRO-1 shards optimizer states: 334 GB. ZeRO-2 adds gradient sharding: 201 GB. ZeRO-3 (FSDP) shards everything: 70 GB — fits in H100. Formula: Memory/GPU = 16Ψ/N_DP. This is a mathematical necessity, not an optimization choice.*

```
ZERO STAGE FORMULAS (N = N_DP):
  None (DDP):   16Ψ / 1       → 70B: 1,120 GB ❌
  Stage 1:      (4 + 12/N)Ψ   → 70B N=16:  334 GB ❌
  Stage 2:      (2 + 14/N)Ψ   → 70B N=16:  201 GB ❌
  Stage 3 (FSDP): 16Ψ/N       → 70B N=16:   70 GB ✅

STAGE 3 COMMUNICATION PER STEP (70B):
  AllGather fwd:      Ψ×2 = 140 GB
  AllGather bwd:      Ψ×2 = 140 GB
  ReduceScatter bwd:  Ψ×2 = 140 GB
  Total: 420 GB per step at 400 GB/s = ~1.05s ← MUST overlap with compute!
```

10.3.3 Ring-AllReduce: How Gradients Move

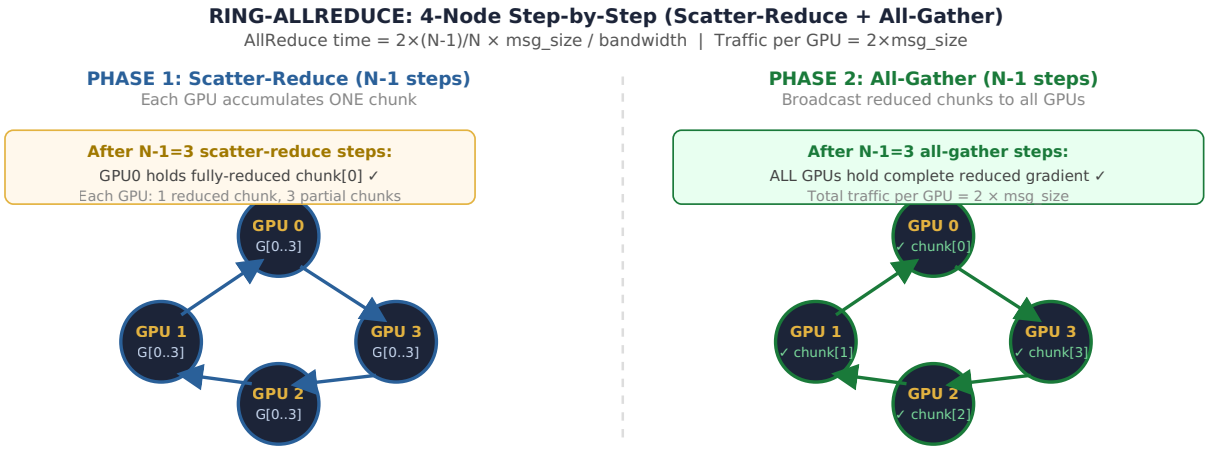


Figure 10.5 *— Ring-AllReduce: 4-node step-by-step. Phase 1 (Scatter-Reduce): Each GPU sends one gradient chunk clockwise while accumulating partial sums. After N−1=3 steps, each GPU holds one fully-reduced chunk. Phase 2 (All-Gather): Broadcast reduced chunks around the ring. After another N−1 steps, every GPU has the complete gradient. Total traffic per GPU = 2×msg_size. Time = 2(N−1)/N × msg_size/bandwidth.*

10.4 — Gradient Communication: Overlap and Bucketing

```
COMPUTE-COMMUNICATION OVERLAP MODEL:

Without overlap: step_time = T_fwd + T_bwd + T_comm + T_opt
With overlap:    step_time = T_fwd + max(T_bwd, T_comm) + T_opt

For 70B on 512×H100 (measured):
  T_fwd = 450 ms, T_bwd = 1,100 ms, T_comm = 1,050 ms, T_opt = 80 ms

  Without overlap: 450 + 1100 + 1050 + 80 = 2,680 ms
  Perfect overlap: 450 + 1100 + 80 = 1,630 ms  → 1.64× faster
  Practical (70%): 450 + 1100 + 315 + 80 = 1,945 ms  → 1.38× faster

MECHANISM: GPU has multiple independent streams
  Stream 0: backward GEMM (uses Tensor Cores)
  Stream 1: NCCL AllReduce (uses NVLink/IB DMA engines)
  Two different hardware units → true simultaneous execution

FSDP OVERLAP CONFIGURATION:
  backward_prefetch=BackwardPrefetch.BACKWARD_PRE   # start AllGather BEFORE layer
  forward_prefetch=True                              # also overlap during forward

GRADIENT BUCKETING (DDP):
  Bucket size 25-50 MB for 70B (too small → 70K tiny NCCL calls; too large → late start)
  model = DistributedDataParallel(model, bucket_cap_mb=50, gradient_as_bucket_view=True)

DIAGNOSIS — is your overlap working?
  nsys profile --trace=cuda,nvtx,nccl python train.py
  GOOD: NCCL bars overlap with CUDA bars (same time window)
  BAD:  NCCL bars appear AFTER all CUDA bars (sequential)
  Target: NCCL fraction of step time <10%
```

10.5 — Fault Tolerance at Scale

```
CLUSTER FAILURE RATE ANALYSIS (~1%/GPU/week):

  GPUs      Failures/week   MTBF
  ──────────────────────────────────
  512       5.1             ~33 hours
  1,024     10.2            ~16 hours
  4,096     40.9            ~4 hours
  16,384    163.8           ~1 hour!

LLaMA-3 405B on 16,384 GPUs for 50 days:
  Expected failures: ~1,148 total (23/day)
  Without fault tolerance: job dies in ~1 hour
  With fault tolerance: 1,148 recovered; run completes

OPTIMAL CHECKPOINT INTERVAL:
  interval = sqrt(2 * T_checkpoint * T_recovery)

  Async checkpoint: T_c = 5s (snapshot to CPU RAM, then async write)
  MTTR: T_r = 600s (10 min)
  interval = sqrt(2 * 5 * 600) = sqrt(6000) = 77.5s ≈ 38 steps at 2s/step

OVERHEAD COMPARISON:
  Synchronous (90s write, 38-step interval):
    overhead = 90 / (38*2 + 90) = 90/166 = 54% ← UNACCEPTABLE

  Async (5s snapshot, write in background):
    overhead = 5 / (38*2 + 5) = 5/81 = 6.2% ← acceptable

HOT-STANDBY NODES: pre-allocate 5% spare nodes
  512-GPU cluster: 4 spare nodes; booted, drivers loaded, container pulled
  Restart target: <10 minutes from failure to training resumption
```

10.6 — Advanced Training Techniques

10.6.2 Activation Checkpointing

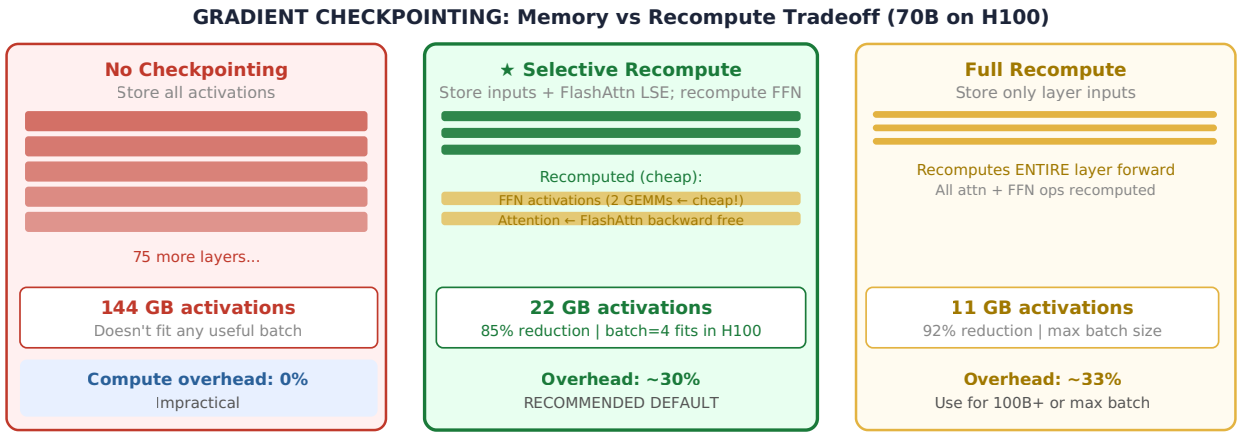


Figure 10.6 *— Gradient checkpointing tradeoff for 70B on H100. No checkpointing: 144 GB activations — impractical. Selective recompute (recommended): store layer inputs and FlashAttention LSE statistics, recompute FFN activations in backward — 22 GB at ~30% overhead. Full recompute: store only one tensor per layer — 11 GB at ~33% overhead for 100B+ models.*

10.6.3 Mixed Precision: The Three-Level Recipe

```
MIXED PRECISION TRAINING (H100, recommended):

Level 1 — BF16 compute:
  All GEMM ops (QKV, FFN, output projections), FlashAttention
  BF16: same exponent range as FP32 → no overflow risk in training
  H100 BF16 GEMM: 989 TFLOPS (8× FP32)

Level 2 — FP32 optimizer:
  FP32 master weights + Adam m,v states
  Why: Adam accumulates tiny gradient updates; BF16 rounds many to zero
  Cost: 12 bytes/param (840 GB for 70B, sharded via ZeRO-3)

Level 3 — FP8 GEMM (H100+ with Transformer Engine):
  FP8 GEMM: 1,979 TFLOPS (2× BF16)
  Dynamic per-tensor scaling automatic via TE
  --fp8-format hybrid: E4M3 forward, E5M2 backward (recommended)
  Quality: <0.3% MMLU degradation (see Chapter 8)
```

10.6.4 Sequence Packing

```
SEQUENCE PACKING — eliminate padding waste:

PROBLEM: web text averages ~600 tokens; seq_len=4096
  Without packing: (4096-600)/4096 = 85% padding waste
  Every GEMM operates on 85% useless tokens

SOLUTION: pack multiple docs per sequence
  [doc_A=2048][doc_B=1024][doc_C=1024] → 100% utilization
  Block-diagonal attention mask: tokens can't attend across doc boundaries

THROUGHPUT GAIN: 15-25% on typical web corpus
  For 70B on 512×H100: +20% = equivalent of 102 extra GPUs
  from pure software optimization (zero hardware cost)
```

10.7 — RLHF Infrastructure

```
RLHF/PPO SYSTEM (70B policy + 70B reference + 13B reward on 512×H100):

Process 1: ROLLOUT WORKERS (40% alloc, ~205 H100s)
  Policy generates responses to prompts
  Needs continuous batching, KV cache, fast decode
  Config: TP=8, ~25 replicas of 70B

Process 2: REWARD SCORER (15% alloc, ~77 H100s)
  13B reward model scores (prompt, response) pairs
  Must match rollout throughput
  Config: TP=4, ~19 replicas of 13B

Process 3: REFERENCE MODEL (25% alloc, ~128 H100s)
  Frozen 70B SFT model computes KL divergence penalty
  Memory mitigation: weight sharing with policy (LoRA delta storage)
    or CPU offload during idle phases

Process 4: PPO POLICY UPDATE (20% alloc, ~102 H100s)
  Gradient updates from PPO objective
  Config: TP=8, PP=2, FSDP for 70B training

DPO vs PPO DECISION:
  Use DPO first: single training process, 4× less infrastructure complexity
  Use PPO when: online feedback loops needed, iterative self-improvement,
                sparse rewards requiring generation-time exploration
```

10.8 — MFU Formula, Monitoring, and Anti-Patterns

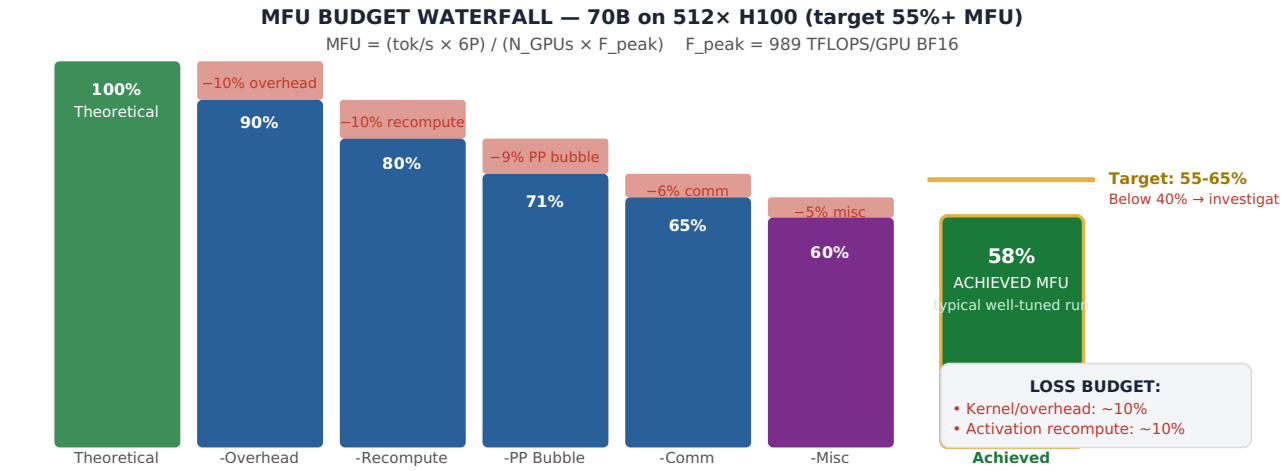


Figure 10.7 *— MFU budget waterfall from 100% to achieved performance. Each bar represents one loss source: kernel/launch overhead (~10%), activation recompute (~10%), pipeline bubble at M=32 (~9%), unabsorbed communication (~6%), and miscellaneous stalls (~5%). A well-tuned run achieves 55–65% MFU. Below 40% requires systematic six-layer investigation.*

```
MFU FORMULA:
  MFU = (tokens_per_second × 6 × P) / (N_GPUs × F_peak)

  P = parameters, F_peak = 989 TFLOPS BF16 per H100
  6 = FLOPs per token per parameter (2 fwd + 4 bwd)

  Example: 70B on 512×H100, measured 500K tok/s:
    MFU = (500,000 × 6 × 70e9) / (512 × 989e12)
        = 210e15 / 506.4e15 = 41.5%  ← below target, investigate!

  TYPICAL MFU RANGES:
    <30%: Critical bottleneck — investigate all 6 layers immediately
    30-45%: Suboptimal — worth tuning (usually comm overlap or bubble)
    45-60%: Good — well-configured
    60-70%: Excellent — near state of practice (Meta LLaMA-3 level)
    >70%: Outstanding — best-in-class systems only
```

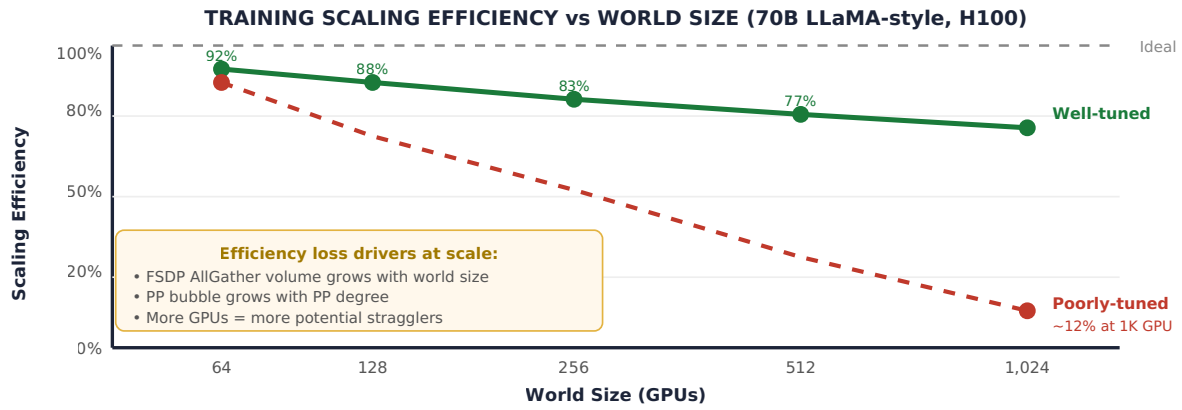


Figure 10.8 *— Training scaling efficiency vs world size. A well-tuned configuration (correct TP/PP placement, FSDP overlap, adequate M) maintains ~83% efficiency at 256 GPUs and ~77% at 1,024. A poorly-tuned configuration (TP across IB, no overlap, small M) can fall to ~12% at 1,024 GPUs — paying for 1,024 GPUs but getting the work of ~123.*

```
EIGHT NUMBERS EVERY TRAINING ENGINEER KNOWS COLD:
  1. 16 bytes/param    — Full Adam memory: 2(wts)+2(grads)+4(m)+4(v)+4(master)
  2. TP ≤ 8            — Max safe TP degree; IB is 12× worse than NVLink for TP
  3. (PP-1)/(PP-1+M)   — Pipeline bubble formula; M ≥ 4×PP for <10%
  4. sqrt(2×Tc×Tr)   — Optimal checkpoint interval
  5. 3× AllReduce     — ZeRO Stage 3 communication (AllGather+AllGather+RS)
  6. ~1%/GPU/week      — GPU failure rate; 512 GPUs → failure every 33 hours
  7. ~30% compute      — Activation recompute overhead (full recompute strategy)
  8. ~20% throughput   — Sequence packing gain on variable-length web corpus
```

10.9 Profiling Cookbook

🔬 **PROFILING COOKBOOK — TRAINING SYSTEM DIAGNOSIS**

```
# ──────────────────────────────────────────
# SYMPTOM: MFU = 28%; GPU SM utilization shows 95% in Nsight Compute
# (These two numbers are NOT contradictory — see Step 2 for why)
# ──────────────────────────────────────────

# STEP 1: Compute true MFU from training log
python3 -c "
tokens_per_sec = 42500   # from training log (global tokens/sec)
P = 30e9                 # model parameters
N = 256                  # GPUs
F_peak = 989e12          # H100 BF16 TFLOPS
MFU = (tokens_per_sec * 6 * P) / (N * F_peak)
print(f'MFU = {MFU:.1%}')
"
# MFU = 28.7%  -- confirmed low. Now find which layer is the bottleneck.

# STEP 2: Why 95% SM util != 95% MFU?
# SM "active" = executing SOMETHING. Could be FP32 CUDA cores (33 TFLOPS)
# not BF16 Tensor Cores (989 TFLOPS). Check TC utilization specifically:
ncu --section ComputeWorkloadAnalysis \
    --kernel-name ".*gemm.*" --launch-skip 10 --launch-count 2 \
    python train.py --steps 3
# sm__pipe_tensor_cycles_active.avg.pct_of_peak: 31%  <— LOW!
# Cause: weight dimensions not aligned to Tensor Core requirements (mult of 16)
# Fix: ensure all hidden/FFN dims are multiples of 64

# STEP 3: Check gradient communication overlap with nsys
nsys profile \
    --trace=cuda,nvtx,nccl \
    --output=/tmp/train_trace \
    --duration=30 \
    python train.py --steps 10
# Open in Nsight Systems GUI. Look for:
# GOOD: NCCL AllReduce timeline rows overlap with CUDA kernel rows
# BAD:  NCCL rows appear entirely AFTER compute rows (sequential!)
# BAD signature: FSDP AllGather completes, then backward starts (not pipelined)

# STEP 4: Verify FSDP overlap is configured correctly
python3 -c "
from torch.distributed.fsdp import FullyShardedDataParallel as FSDP
from torch.distributed.fsdp import BackwardPrefetch
# Check your model config has these set:
print('Required FSDP overlap config:')
print('  backward_prefetch=BackwardPrefetch.BACKWARD_PRE')
```

```
print('  forward_prefetch=True')
print('  limit_all_gathers=True  # memory safety')
"

# STEP 5: Check pipeline bubble fraction
python3 -c "
PP = 4    # your pipeline stages
M  = 8    # micro-batches per step (grad_accum * DP / PP)
bubble = (PP-1) / (PP-1 + M)
print(f'Bubble fraction: {bubble:.1%}')
target_M = (PP-1) * 10 - (PP-1) + 1  # for <10% bubble: M >= 4*PP
print(f'Target M for <10% bubble: {4*PP}')
print(f'Action: set gradient_accumulation_steps = {4*PP * PP // max(1,16)}')
"
# Bubble fraction: 27.3%  — VERY HIGH
# Target M for <10% bubble: 16
# Action: increase gradient_accumulation_steps

# STEP 6: Check data pipeline is not starving the GPU
watch -n 0.5 "nvidia-smi --query-gpu=utilization.gpu,utilization.memory \
    --format=csv,noheader"
# If GPU util drops to 0% periodically between steps: DataLoader starvation
# Fix: increase num_workers, use larger shard files, enable pin_memory=True

# STEP 7: Diagnose silent data corruption (run periodically)
python3 -c "
# In your training loop, add gradient norm monitoring:
grad_norm = torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
if torch.isnan(grad_norm) or grad_norm > 5.0:
    print(f'WARNING: gradient spike at step {step}: {grad_norm:.2f}')
    # Skip optimizer step; increment spike counter
    # If >3 spikes in 100 steps: trigger checkpoint rollback
"
# ECC error check: nvidia-smi --query-gpu=ecc.errors.uncorrected.volatile.total
# If XID 79 (DBE): immediately drain that GPU; checkpoint current state
```

Worked Example: Configuring 70B from Scratch on 512× H100

WORKED EXAMPLE 10.1 — COMPLETE 70B TRAINING CONFIGURATION: MEMORY MATH, PARALLELISM, AND EXPECTED MFU

Step 1: Memory Sizing

```
MODEL: LLaMA-3 70B (P=70B, L=80, d_model=8192, GQA 8 KV heads)
HARDWARE: 512× H100 SXM5, 80 GB HBM, NVLink intra-node, IB NDR 400G inter

Without sharding: 70B × 16 bytes/param = 1,120 GB → need ZeRO Stage 3

Minimum N_DP for ZeRO-3 to fit:
  16 × 70B / N_DP ≤ 75 GB (leaving 5 GB headroom for activations and misc)
  N_DP ≥ 16 × 70 / 75 = 14.9 → N_DP = 16

ZeRO-3 states at N_DP=16:
  16 × 70B / 16 = 70.0 GB

Activations (selective recompute, micro-batch=2, seq=4096):
  ~11 GB (from Chapter 2 derivation; full recompute = ~5.5 GB)

Total per GPU: 70.0 + 11.0 = 81.0 GB ← slightly over! Fix: micro-batch=1 → ~6 GB activations
  OR: enable SP (saves ~8 GB activation) → micro-batch=2 works ✅

With sequence parallelism (TP=8):
  Activation savings: ~8 GB → total = 70.0 + 3.0 = 73.0 GB ✅ fits with headroom
```

Step 2: Parallelism Configuration

```
STEP 2A: TP = 8 (full NVLink node, mandatory)
  AllReduce latency/layer: 268 MB / 600 GB/s = 0.45 ms
  80 layers × 2 AllReduces × 0.45 ms = 72 ms/step ✅

STEP 2B: PP = 4 (assigns 20 layers per stage)
  P2P activation volume/stage: 2 × 4096 × 8192 × 2 = 134 MB
  IB latency: 134 MB / 50 GB/s = 2.7 ms/hop ✅ (acceptable; only 3 hops/step)

STEP 2C: CP = 1 (seq_len = 4096 < 32K threshold)

STEP 2D: DP = 512 / (8 × 4 × 1) = 16 FSDP replicas
  Verify: 8 × 4 × 1 × 16 = 512 ✅

STEP 2E: Gradient accumulation for bubble control
  Target M ≥ 4 × PP = 16
  grad_accum = 8 → M = 8 × 16 / 4 = 32
  Bubble = (4-1)/(4-1+32) = 3/35 = 8.6% ✅

STEP 2F: Effective batch size
  effective_batch = 2 × 8 × 16 = 256 sequences × 4096 = 1,048,576 tokens
  Chinchilla-optimal range: 2M–4M tokens → consider doubling grad_accum to 16
```

Step 3: Expected MFU Estimation

```
THEORETICAL MAXIMUM THROUGHPUT (100% MFU):
  tokens/s = N × F_peak / (6 × P) = 512 × 989e12 / (6 × 70e9)
           = 506.4e15 / 420e9 = 1,205,714 tok/s

APPLY LOSS FACTORS:
  PP bubble (8.6%):            × 0.914
  Activation recompute (~10%): × 0.90
  Comm overhead (not hidden):  × 0.92
  Kernel/launch overhead:      × 0.95

Expected: 1,205,714 × 0.914 × 0.90 × 0.92 × 0.95
        = 1,205,714 × 0.721 = 869,320 tok/s

Expected MFU: 869,320 / 1,205,714 = 72.1% (optimistic; practical ~58–65%)

VERIFY AT RUNTIME (first thing to check after launch):
  MFU = (measured_tok/s × 6 × 70e9) / (512 × 989e12)
  At 700K tok/s: MFU = (700K × 420e9) / 506.4e15 = 58.1% ✅ good
  At 300K tok/s: MFU = (300K × 420e9) / 506.4e15 = 24.9% ❌ investigate!
```

Step 4: Cost Projection

```
TRAINING RUN COST (300B token pretraining budget, owned hardware):

Hardware cost: 512 × H100 at $2/GPU-hr fully loaded
Token target: 300 billion tokens (Chinchilla-optimal for 70B)

At 700K tok/s (58% MFU):
  Time = 300e9 / 700,000 = 428,571 seconds = 119 hours = 5.0 days
  Cost = 512 GPUs × 119 hours × $2/hr = $121,856

At 400K tok/s (33% MFU, poorly tuned):
  Time = 300e9 / 400,000 = 750,000 seconds = 208 hours = 8.7 days
  Cost = 512 × 208 × $2 = $212,992

DIFFERENCE: $91,136 saved per training run from proper configuration.
At 4 training runs/quarter: $364K/year savings from setup discipline alone.
```

CHAPTER 10 QUICK REFERENCE — KEY FORMULAS AND NUMBERS

Formula	Definition	70B on 512× H100
MFU = (tok/s × 6P) / (N × F_peak)	Model FLOPs Utilization	Target 55–65%; F_peak=989 TFLOPS BF16
= (PP−1) / (PP−1+M)	Pipeline idle fraction	PP=4, M=32 → 8.6%; need M ≥ 4×PP
ZeRO-3 = 16Ψ / N_DP	Per-GPU memory, full sharding	16×70B/16 = 70 GB; N_DP ≥ 15 needed
= 2(N−1)/N × M / BW	Ring-AllReduce latency	TP: 268 MB, NVLink 600 GB/s → 0.45 ms
Eff. batch = mbs × accum × DP	Global tokens per optimizer step	2×8×16=256 seq × 4096 = 1M tokens
= sqrt(2×Tc×Tr)	Optimal checkpoint interval	Async Tc=5s, Tr=600s → 77.5s ≈ 38 steps
FSDP comm = 3Ψ × 2 bytes	Communication per step	3×140 GB = 420 GB; ~1.05s at 400 GB/s

Parameter	70B, 512× H100	Rule / Constraint
TP	8	Must ≤ GPUs per NVLink node; NEVER cross IB
PP	4	20 layers/stage; bubble = (PP−1)/(PP−1+M)
DP	16	Total/(TP×PP×CP); enables ZeRO-3 via FSDP
CP	1	Enable only for seq > 32K tokens
Grad accum	8	M = accum×DP/PP = 32 → bubble = 8.6%
ZeRO-3 states/GPU	70 GB	16×70B/16 = 70 GB; 10 GB headroom for activations
TP AllReduce latency	0.45 ms/layer	268 MB at NVLink 600 GB/s vs 5.4 ms at IB
FSDP total comm/step	420 GB	3×Ψ×2; ~1.05s; MUST overlap with backward
GPU failure rate	~1%/GPU/week	512 GPUs → failure every ~33 hours
Achieved MFU (typical)	55–65%	Meta LLaMA-3 runs achieved ~65%
Sequence packing gain	~20%	Variable-length web corpus; zero hardware cost
Async checkpoint overhead	~6%	5s snapshot / (38×2+5)s = 6.2%

Key Takeaways

CORE CONCEPTS

- Training at scale is a six-layer stack. The bottleneck shifts unpredictably between GPU hardware, NCCL communication, framework overhead, data pipeline, and storage. Profile all six layers before optimizing any one — the most common mistake is kernel-tuning when the data pipeline is actually starving the GPU.
- ZeRO Stage 3 (FSDP) is a mathematical necessity, not an optimization. The formula 16Ψ/N_DP determines per-GPU memory after full sharding. For 70B with N_DP=16: exactly 70 GB — fitting a H100 by design. Without it, 70B requires 1,120 GB per GPU — 14× capacity. Every model above ~5B parameters on typical GPU counts requires Stage 3.
- The pipeline bubble formula bubble = (PP−1)/(PP−1+M) governs how much cluster compute is idle during pipeline fill and drain. Set M ≥ 4×PP. The interleaved schedule with V virtual stages reduces the formula denominator to V×M — the most reliable way to reduce bubbles without reducing PP degree.
- TP must never cross InfiniBand. AllReduce latency on NVLink is 0.45 ms per layer; on IB it is 5.4 ms — a 12× difference. With 80 layers × 2 AllReduces per step, this means 72 ms vs 864 ms per step from TP communication alone. This single anti-pattern can cut MFU by 50%.

COST & PRODUCTIVITY IMPACT

- Sequence packing delivers 15–25% throughput improvement on variable-length web corpora at zero hardware cost. On a 512-GPU cluster at $2/GPU-hr, a 20% throughput gain saves $90K per 5-day training run. This is a two-line configuration change: enable packing, set correct attention mask for document boundaries.
- Async checkpointing reduces overhead from 54% (synchronous, 90s write every 38 steps) to 6.2% (5s snapshot, then background write). Applied to a 30-day run: that is 13 days of recovered training time. At $2/GPU-hr on 512 GPUs, 13 days = $748,800 saved per long run — from a one-day engineering change.
- Proper 4D configuration (vs poorly-tuned) saves $91K per 300B-token pretraining run at $2/GPU-hr on 512 GPUs (58% vs 33% MFU). Four runs per quarter = $364K/year purely from configuration discipline. No hardware purchase required.

PRINCIPAL-LEVEL INSIGHT

- 95% SM utilization is consistent with 28% MFU. The SM is active but executing FP32 CUDA core ops instead of BF16 Tensor Core ops — a 30× throughput gap (33 TFLOPS vs 989 TFLOPS). The revealing metric is `sm__pipe_tensor_cycles_active.avg.pct_of_peak` in Nsight Compute, not SM utilization. Misaligned weight dimensions (not multiples of 16) silently trigger this fallback with no visible error.
- The optimal checkpoint interval is deterministic: sqrt(2×Tc×Tr). With async snapshot Tc=5s and MTTR Tr=600s, the optimal is ~77.5 seconds (~38 steps at 2s/step). Teams that checkpoint every 100 steps on a 2s/step run are not at the optimum — they are accepting more expected recompute loss on failure than necessary. Always derive the interval from the formula, not from "round numbers feel safe."
- Gradient communication overlap is the single highest-ROI training optimization that teams consistently misconfigure. The failure modes are subtle: `NCCL_LAUNCH_MODE=GROUP` serializes streams; missing `backward_prefetch=BACKWARD_PRE` delays AllGather start; bucket sizes under 5 MB cause tens of thousands of tiny NCCL calls instead of hundreds of large ones. The Nsight Systems signature of a working overlap is NCCL bars and compute bars occurring at the same wall-clock time — anything else means overlap has failed.

Review Questions

FOUNDATIONAL Build the mental model — no math required

1. Explain why Tensor Parallelism must always stay within a NVLink domain and must never cross InfiniBand. What is the quantitative consequence for a 70B model if TP is placed across IB instead of NVLink? Express your answer in ms/step.
2. A colleague proposes using ZeRO Stage 1 for a 70B model on 32 H100s instead of Stage 3, arguing "Stage 3 has too much communication overhead." Using the memory formula, show numerically why Stage 1 is physically impossible on this hardware, then explain why Stage 3's communication overhead is actually acceptable when overlapped correctly.
3. A training run shows 95% GPU SM utilization but only 28% MFU. These numbers appear contradictory. Explain precisely how both can be simultaneously true, name the specific Nsight Compute metric that reveals the true bottleneck, and describe what it means when that metric reads 31%.

INTERMEDIATE Apply and calculate — production context

1. You are configuring training for a 30B model (48 layers, d_model=8192) on 256× H100s (32 nodes, 8 GPUs/node). (a) What is TP? (b) Calculate ZeRO-3 memory/GPU at N_DP=4, 8, and 16. Which is the minimum that fits? (c) Choose PP and verify layers/stage ≥ 10. (d) With your chosen PP and minimum viable DP, calculate the gradient accumulation steps needed for pipeline bubble <10%. (e) Verify: TP×PP×DP = 256.
2. An Nsight Systems trace shows a 70B training step on 512×H100: forward=450ms, backward=900ms, NCCL AllGather+ReduceScatter=480ms (entirely sequential with backward), optimizer=80ms, total=1,910ms. (a) Compute current MFU at 85,000 tok/s. (b) What is the theoretical minimum step time with perfect overlap? (c) List the exact FSDP configuration parameters that enable overlap. (d) What is the expected MFU improvement if 80% overlap efficiency is achieved?
3. A 4,096-GPU training cluster fails at 1%/GPU/week. Synchronous checkpoints take 120 seconds to write and are triggered every 100 steps at 1.5s/step. (a) What is the mean time between failures? (b) Using the optimal checkpoint interval formula with async Tc=8s and MTTR Tr=600s, what is the optimal interval in steps? (c) What is the checkpoint overhead percentage under both synchronous and async approaches? (d) How many total GPU-hours per week are wasted by synchronous checkpointing vs async?

PRINCIPAL LEVEL Design, decide, and defend — architecture reviews

1. **175B configuration on 2,048× H100:** 175B parameters, 96 layers, d_model=12,288, on 256 nodes (8 GPUs/node, NVLink intra, IB NDR 400G inter). (a) Derive TP, PP, DP, and CP values. Show the complete ZeRO-3 memory calculation including activations at batch=4, seq=4096 — prove your configuration fits in 80 GB. (b) Compute TP AllReduce latency for your TP choice; verify it's acceptable on NVLink. (c) Calculate pipeline bubble fraction with your PP. Determine gradient accumulation steps for bubble <8%. (d) If the run requires 128K-token sequences for long-context capability, what changes and why? Show the new memory calculation with CP enabled.

2. **Fault tolerance architecture for production 8,192-GPU cluster:** Design a fault tolerance system achieving <5% throughput overhead from checkpointing and <10-minute mean recovery time. (a) Calculate expected failures/week and mean time between failures at 1%/GPU/week. (b) Using the optimal checkpoint interval formula with async Tc=6s and Tr=8 minutes, compute the optimal interval in steps at 1.5s/step. (c) Show the checkpoint overhead percentage at this interval. (d) Design the four-layer silent data corruption detection hierarchy — specify the exact metric, action threshold, and automated response for each layer. (e) Size the hot-standby node pool for 99.5% uptime. Show your math using the MTBF from part (a) and the 10-minute recovery target.

3. **End-to-end MFU investigation:** A 30B model on 256× H100s achieves 28% MFU. SM utilization is 95%. Construct a complete systematic debugging plan covering all six training stack layers. For each layer: (a) specify the diagnostic tool and exact metric to examine, (b) describe what a positive finding looks like in that tool's output, and (c) give the specific remediation. Then: (d) explain how 95% SM utilization is consistent with 28% MFU — which specific profiler counter distinguishes the two, and what does it reveal? (e) Assume the root cause is that gradient communication is entirely sequential with the backward pass. Calculate the current step time breakdown and the theoretical improvement from achieving 80% overlap efficiency. (f) List all six anti-patterns from the chapter that could each independently cause 28% MFU, and rank them by frequency of occurrence in production systems.
