

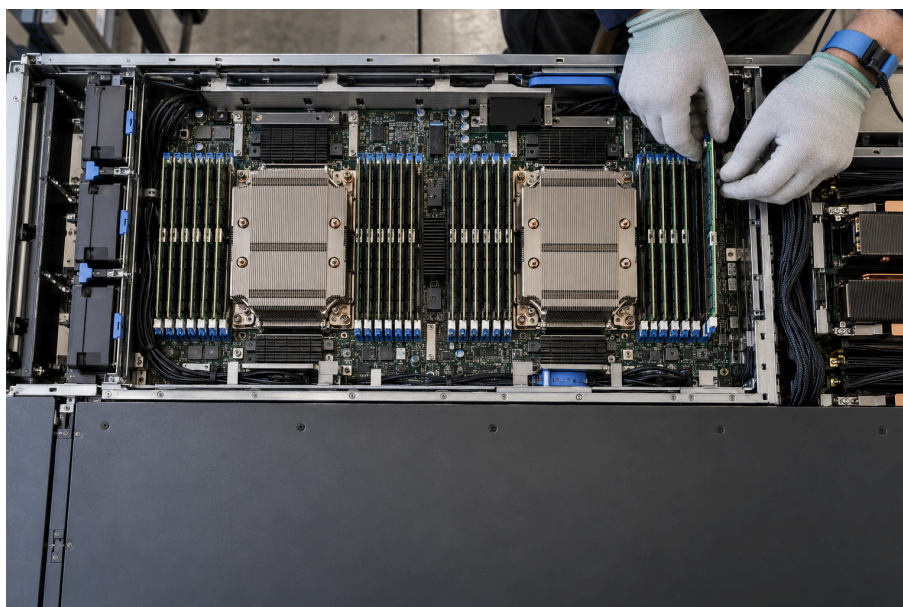


Host CPU and System RAM Sizing for GPU Servers

Plan and build private AI compute with GPU Smith. We connect workload requirements with GPU hardware, networking, deployment and operating decisions so your infrastructure fits the work it must perform.

Published by GPU Smith · 2026-10-29 · how many cpu cores per gpu · system ram per gpu for llm server · cpu requirements for gpu inference server vllm · dgx h100 cpu cores memory · grace vs x86 host for gpu server · dram vs hbm ratio ai server

Original article: <https://gpusmith.com/articles/en/host-cpu-and-system-memory-sizing-for-gpu-servers>



In this article

- 1. The short answer, and why it is usually guessed
- 2. Reference ratios from NVIDIA systems
- 3. The inference floor: vLLM's process model
- 4. Where inference CPU time goes, and how to measure it
- 5. Training hosts: data loaders, pinned memory, checkpoint staging
- 6. System memory sizing: a worked inference example
- 7. Memory channels and bandwidth
- 8. PCIe lanes and topology
- 9. Grace versus x86 hosts
- 10. NUMA and binding
- 11. Cost trade-offs
- 12. Sizing worksheet
- 13. Sources

The short answer, and why it is usually guessed

GPU server quotes get scrutinised for the accelerators and filled in with the vendor's default CPU and DIMMs, leaving GPUs idling behind a starved host or a terabyte of DRAM nobody planned to use. There is no published law for host sizing, but there are two anchors. NVIDIA's own 8-GPU DGX systems ship with 14 to 16 CPU cores and about 256 GB of DRAM per GPU. vLLM documents a hard floor for inference: at least 2 + N physical cores for N GPUs, more with data parallelism. Size cores upward from that floor using measured CPU work (tokenization, multimodal preprocessing, data loading). Size RAM as a sum of named consumers: page cache for weights, a KV-cache offload tier, pinned buffers, checkpoint staging, plus headroom. The DGX ratios are one vendor's reference points, not requirements.

Reference ratios from NVIDIA systems

Specs are cited; the per-GPU and DRAM:HBM columns are our arithmetic.

System	Host CPUs (cited)	Cores (total)	System RAM (cited)	GPU memory (cited)	Cores/ GPU	DRAM/ GPU	DRAM: HBM
DGX H100	2 x Xeon 8480C, 56 cores each	112	2 TB, 32 DIMMs	640 GB (8 x H100)	14	256 GB	3.2
DGX H200	same as DGX H100	112	2 TB, 32 DIMMs	1,128 GB (8 x H200)	14	256 GB	1.8
DGX B200	2 x Xeon 8570, 56 cores each	112	2 TB, upgradable to 4 TB	1,440 GB (8 x B200)	14	256 GB	1.4
DGX B300	2 x Xeon 6776P (64 cores each per Intel)	128	2 TB default, up to 4 TB	8 x 288 GB = 2.3 TB (product page: 2.1 TB)	16	256 GB	0.9

System	Host CPUs (cited)	Cores (total)	System RAM (cited)	GPU memory (cited)	Cores/GPU	DRAM/GPU	DRAM:HBM
GB200 NVL72 (per superchip)	1 Grace, 72 Neoverse V2 cores	72	up to 480 GB LPDDR5X	372 GB HBM3E (2 GPUs)	36	up to 240 GB	1.3

Sources: [DGX H100/H200 user guide](#); [DGX B200 user guide](#); [DGX B300 user guide](#), which names the CPUs but not their core count, taken from [Intel's 6776P specification page](#) (64 cores, 8 memory channels, 88 PCIe lanes, 350 W); [GB200 NVL72 page](#) ("1 Grace CPU | 2 Blackwell GPU" per superchip).

DRAM/GPU treats 2 TB as 2,048 GB (the H100/H200 and B200 guides specify 32 DIMMs, so 64 GB each by our arithmetic; the B300 guide gives no DIMM count, only 128 GB DIMMs for the 4 TB upgrade). DRAM:HBM is $2,048 / 640 = 3.2$ for H100 and $2,048 / 2,304 = 0.89$ for B300.

1. **Cores per GPU have barely moved:** 14 to 16 across three DGX generations, while GPU FLOPS rose sharply.
2. **DRAM per GPU is flat at about 256 GB in the default configs, so DRAM:HBM has collapsed** from 3.2x on H100 to under 1x on B300. At the default 2 TB, a B300 host cannot hold one full copy of its HBM. Designs that mirror HBM contents into DRAM (KV offload, checkpoint staging) should price the 4 TB option.
3. **Grace changes the shape:** 36 cores and up to 240 GB per GPU on GB200.

The inference floor: vLLM's process model

vLLM's [optimization and tuning guide](#) (v0.32.0, released 2026-10-18) lists one API server process (which handles "HTTP requests, tokenization, and input processing"), one engine core process (which "runs the scheduler and coordinates GPU workers"), and one worker per GPU. "The minimum is 2 + N physical cores", and "Using fewer physical CPU cores than processes will cause contention and significantly degrade throughput and latency."

- **Physical cores, not vCPUs.** With hyperthreading "1 vCPU = 1 hyperthread = 1/2 physical CPU core", so the minimum becomes $2 \times (2 + N)$ vCPUs. A request of 10 CPUs for an 8-GPU pod is half the floor on an SMT host.
- **Data parallelism adds processes:** "Minimum physical cores = $A + DP + N + (1 \text{ if } DP > 1 \text{ else } 0)$ ", where A is the API server count (default DP).

Layout (8 GPUs)	Formula	Physical-core floor	vCPU floor with SMT
One TP=8 replica	$2 + 8$	10	20
Two TP=4 replicas, DP=2	$2 + 2 + 8 + 1$	13	26
Four TP=2 replicas, DP=4	$4 + 4 + 8 + 1$	17	34

A DGX-class host has 112 to 128 cores, so the floor uses 10 to 15 percent of them. The floor is where vLLM says performance falls apart, not a target; everything above it is budget for CPU work that scales with traffic.

Where inference CPU time goes, and how to measure it

The same vLLM guide names the scaling CPU work: "tokenization, chat template rendering, and multi-modal data loading all run on CPU", and notes that more cores "and even the clock speed can significantly improve end-to-end performance." For multimodal requests, "By default, 8 CPU threads are used in each API server to load media items", and the multimodal processor cache defaults to 4 GiB (`mm_processor_cache_gb`), with total cache memory of `mm_processor_cache_gb * data_parallel_size`.

We do not quote a "tokens per second per core" figure for tokenization: it depends on tokenizer, prompt lengths, chat template and image sizes, and we found no source we would stand behind. Measure it:

1. Load-test at the target request rate with production-like prompts and media.
2. Watch per-process CPU (`top -H` or `pidstat -t`) for the API server and engine core. An API server pinned at 100 percent of a core while GPU utilization sags is the pattern vLLM describes: "CPU contention may be the bottleneck."
3. Add API servers (the guide's scale-out pattern) and re-measure. If throughput rises, input processing was the bottleneck. Scale-out disables multimodal IPC caching, and `VLLM_MEDIA_LOADING_THREAD_COUNT` may need lowering to avoid CPU exhaustion.
4. Record cores in use at target load and add burst headroom.

Image-heavy and long-document workloads are where 14 cores per GPU earn their cost.

Training hosts: data loaders, pinned memory, checkpoint staging

Each PyTorch DataLoader worker is a separate process, so 8 GPUs with 8 workers each is 64 decode-and-augment processes plus the ranks. Image and video augmentation can consume every core a DGX has; pre-tokenized text needs few. The signal to watch is step time stretching while loader workers are saturated.

Pinned memory is the second consumer. The [PyTorch pinned-memory tutorial](#) explains that pageable data must first be staged into "a page-locked copy", and warns that large pinned allocations "can monopolize a substantial portion of RAM" and "cannot be swapped out to disk." Count them.

Checkpoint staging is the third and usually the largest. The [PyTorch async checkpoint recipe](#) says async checkpointing copies models "into internal CPU-buffers" and "does raise CPU memory by a factor of `checkpoint_size_per_rank X number_of_ranks`", and that with pinned buffers "the same peak memory pressure being sustained" between steps.

Worked example: checkpoint staging for a 70B training run (hypothetical).

- Assumption: mixed-precision Adam. The [ZeRO paper](#) counts 2 (FP16 parameters) + 2 (gradients) + 12 (FP32 copy, momentum, variance) = 16 bytes per parameter. Checkpoints skip gradients: assume 14 bytes.
- Full checkpoint: $70 \times 10^9 \times 14 \text{ bytes} = 980 \text{ GB}$.
- Sharded across 8 nodes (64 GPUs), each node stages $980 / 8 = 122.5 \text{ GB}$.

- On a 2 TB node that is about 6 percent of RAM. Unsharded state would stage 980 GB per node, nearly half of RAM, held continuously if buffers persist. Check your real checkpoint size before trusting the 14-byte assumption.

System memory sizing: a worked inference example

Scenario (hypothetical). One 8 x H100 80 GB node serving Qwen2.5-72B-Instruct in BF16 on vLLM as two TP=4 replicas (DP=2), with a CPU KV tier for prefix reuse.

Step 1: model facts. The [Qwen2.5-72B-Instruct config](#) gives 80 layers, 8 KV heads, hidden size 8,192 and 64 attention heads (head dim $8,192 / 64 = 128$). Its [safetensors index](#) reports `total_size` 145,412,407,296 bytes, so weights are 145.4 GB.

Step 2: KV bytes per token (BF16). $2 \text{ (K and V)} \times 80 \text{ layers} \times 8 \text{ heads} \times 128 \text{ dims} \times 2 \text{ bytes} = 327,680 \text{ bytes}$, about 0.33 MB per token.

Step 3: GPU-side KV, for scale. Assume 90 percent of 4 x 80 GB usable: $288 - 145.4 = 142.6 \text{ GB}$, or $142.6 \times 10^9 / 327,680 = \text{about } 435,000 \text{ tokens per replica}$ (an upper bound; activations ignored).

Step 4: CPU KV tier. vLLM's [KV offloading guide](#) sets the tier with `cpu_bytes_to_use`, "Total bytes of host memory reserved for the CPU tier across all workers (not per-worker)", and copies blocks "into pinned host memory". Target one million cached tokens per replica, a bit over twice the GPU tier: $10^6 \times 327,680 = 327.7 \text{ GB per replica}$, 655.4 GB for both. This is pinned memory.

Step 5: weight staging and page cache. Loading from local NVMe typically goes through the Linux page cache; keeping the shards cached speeds restarts. Both replicas read the same files, so count 145.4 GB once.

Step 6: everything else. OS, runtime, Python heaps, CUDA host allocations, agents. We have no sourced figure; measure a loaded, idle node and call it X.

Total: $655.4 + 145.4 + X = 800.8 \text{ GB} + X$.

Decision. 1 TB leaves about 200 GB for X and headroom: workable but tight, since pinned pages cannot be reclaimed and page cache can. 2 TB allows doubling the KV tier or caching a second model. Without the KV tier the node needs about 145 GB plus X, and 512 GB is plenty. The KV tier, not GPU count, drives the decision.

Transfer cost check: a 32,768-token cached prefix is $32,768 \times 327,680 = 10.7 \text{ GB}$, or 2.7 GB per GPU under TP=4. The [H100 spec page](#) rates PCIe Gen5 at 128 GB/s (both directions combined). At an assumed 50 GB/s achieved one-way, reloading takes about 54 ms per GPU. Whether that beats recomputing the prefix depends on your prefill speed; measure it.

Memory channels and bandwidth

The Xeon 6776P supports 8 DDR5 channels at 6,400 MT/s ([Intel](#)). Theoretical peak per socket is $8 \times 6,400 \times 10^6 \times 8 \text{ bytes} = 409.6 \text{ GB/s}$. Populate every channel on both sockets with identical DIMMs; empty channels lose bandwidth that KV offload and checkpoint staging use. Check the vendor's population guide for speed at two DIMMs per channel before choosing 16 large or 32 small DIMMs.

PCIe lanes and topology

Lane arithmetic explains why 8-GPU servers do not attach everything directly to the CPUs. Eight GPUs at x16 is 128 lanes; eight 400G NICs at x16 is another 128; eight NVMe drives at x4 is 32. That totals 288 lanes, against $2 \times 88 = 176$ on a dual 6776P host. Devices therefore share upstream links through PCIe switches, so which GPU sits under which switch and socket matters for KV offload. Get the block diagram before ordering, and verify negotiated link width and speed with our [PCIe link troubleshooting](#) guide.

Grace versus x86 hosts

The [Grace CPU Superchip](#) page lists 144 Neoverse V2 cores, LPDDR5X options of 240, 480 and 960 GB, bandwidth "Up to 1024 GB/s for 240GB, 480GB" (768 GB/s for 960 GB), and NVLink-C2C "Up to 900 GB/s". In GB200 each 72-core Grace serves two GPUs with up to 480 GB LPDDR5X at up to 512 GB/s ([GB200 NVL72](#)).

- **For Grace:** more cores per GPU, NVLink-C2C rather than PCIe between CPU and GPUs (per the GB200 page), and up to 512 GB/s of memory bandwidth per Grace against 409.6 GB/s theoretical for a 6776P socket.
- **Against Grace:** on-module memory fixes capacity at purchase (480 GB per GB200 superchip at most), and every host-side dependency needs a validated Arm build.
- **For x86:** up to 4 TB on DGX B200/B300, with on-site expansion using 128 GB DIMMs per the B300 guide.

Needing more than about 240 GB of host memory per GPU, for a large KV tier say, points to x86 with 4 TB. NVIDIA's next host CPU is covered in our [Vera CPU specs](#) article.

NUMA and binding

Two sockets means two NUMA domains; a KV tier allocated on the wrong socket crosses the inter-socket link on every transfer. Binding is covered in [GPU/CPU/NIC affinity and rank binding](#); on Kubernetes, see [pod-level resource managers and NUMA](#) for making CPU and memory requests NUMA-aligned. For sizing: budget cores and RAM per socket, not per node.

Cost trade-offs

GPU idle time is paid at the GPU rate, so weigh the quoted price step to a larger CPU SKU against even a few percent of GPU utilization lost. Do not trim cores below measured need plus headroom, but do not buy 4 TB by default either: DRAM without a named consumer is waste. Reasonable savings: fewer, higher-clocked cores for text-only inference, and 1 TB instead of 2 TB when there is no KV tier and checkpoints are sharded. Get DRAM pricing at order time; it has been volatile.

Sizing worksheet

Input	Example value	Feeds
GPUs per node (N)	8	core floor
Replicas (DP) and API servers (A)	2, 2	core floor
Measured peak cores for input processing / data loading	measure	cores
Distinct models cached on local disk	1 x 145.4 GB	RAM: page cache
KV offload target (tokens x bytes/token)	2 x 10 ⁶ x 327,680 B	RAM: pinned
Checkpoint staging per node	0 (inference)	RAM: pinned or pageable
Multimodal processor cache	4 GiB x DP, if multimodal	RAM
OS, runtime and agents (X)	measure	RAM

Outputs. Cores per socket = (vLLM floor + measured preprocessing + data loaders) / 2, plus burst headroom, doubled for SMT vCPUs. RAM = page cache + KV tier + staging + caches + X + headroom, rounded up to an evenly populated channel configuration. Compare with the 14 to 16 cores and about 256 GB per GPU reference, and write down why you differ.

GPU Smith helps teams run this kind of host sizing against their own traffic before a purchase order is signed.

Sources

1. [DGX H100/H200 User Guide: Introduction](#), NVIDIA (accessed 2026-10-28)
2. [DGX B200 User Guide: Introduction](#), NVIDIA (accessed 2026-10-28)
3. [DGX B300 User Guide: Introduction](#), NVIDIA (accessed 2026-10-28)
4. [Intel Xeon 6776P Processor specifications](#), Intel (accessed 2026-10-28)
5. [NVIDIA GB200 NVL72](#), NVIDIA (accessed 2026-10-28)
6. [NVIDIA Grace CPU Superchip](#), NVIDIA (accessed 2026-10-28)
7. [NVIDIA H100 Tensor Core GPU](#), NVIDIA (accessed 2026-10-28)
8. [Optimization and Tuning](#), vLLM documentation, v0.32.0 (accessed 2026-10-28)
9. [KV Offloading Usage Guide](#), vLLM documentation, v0.32.0 (accessed 2026-10-28)
10. [A guide on good usage of non_blocking and pin_memory\(\) in PyTorch](#), PyTorch (accessed 2026-10-28)
11. [Asynchronous Saving with Distributed Checkpoint \(DCP\)](#), PyTorch (accessed 2026-10-28)
12. [ZeRO: Memory Optimizations Toward Training Trillion Parameter Models](#), Rajbhandari et al., arXiv, 2019 (accessed 2026-10-28)
13. [Qwen2.5-72B-Instruct config.json](#), Qwen on Hugging Face (accessed 2026-10-28)
14. [Qwen2.5-72B-Instruct model.safetensors.index.json](#), Qwen on Hugging Face (accessed 2026-10-28)

Source links

Links cited in this article, in order of first appearance. Full addresses are provided for readers using extracted text.

1. <https://docs.nvidia.com/dgx/dgxm100-user-guide/introduction-to-dgxm100.html>
2. <https://docs.nvidia.com/dgx/dgxm200-user-guide/introduction-to-dgxm200.html>
3. <https://docs.nvidia.com/dgx/dgxm300-user-guide/introduction-to-dgxm300.html>
4. <https://www.intel.com/content/www/us/en/products/sku/243691/intel-xeon-6776p-processor-336m-cache-2-30-ghz/specifications.html>
5. <https://www.nvidia.com/en-us/data-center/gb200-nvl72>
6. <https://docs.vllm.ai/en/v0.32.0/configuration/optimization>
7. https://docs.pytorch.org/tutorials/intermediate/pinmem_nonblock.html
8. https://docs.pytorch.org/tutorials/recipes/distributed_async_checkpoint_recipe.html
9. <https://arxiv.org/abs/1910.02054>
10. <https://huggingface.co/Qwen/Qwen2.5-72B-Instruct/blob/main/config.json>
11. <https://huggingface.co/Qwen/Qwen2.5-72B-Instruct/blob/main/model.safetensors.index.json>
12. https://docs.vllm.ai/en/v0.32.0/features/kv_offloading_usage
13. <https://www.nvidia.com/en-us/data-center/h100>
14. <https://www.nvidia.com/en-us/data-center/grace-cpu-superchip>

THE PUBLISHER

About GPU Smith

GPU Smith provides independent engineering and research for private AI infrastructure. We help technical buyers and operators reason about GPU workload sizing, hardware selection, procurement, deployment and inference operations, from the compute system to the networking, power and cooling requirements around it.

Start with the workload

Useful infrastructure decisions begin with the models, data, throughput, latency and operating constraints a team actually has. GPU Smith helps connect those requirements with system design and deployment choices. The goal is a privately controlled AI environment whose capacity and operational demands are understood before a purchasing decision.

Hardware and supplier research

Our public reference library covers GPUs, complete systems and networking components. The server-vendor directory and manufacturing research help teams investigate suppliers, compare options and follow sources behind technical claims. We distinguish manufacturer specifications from measured benchmarks and advertised capabilities from tested configurations. Reference pages are research resources, not a live inventory listing or a binding equipment quote.

Deployment and operations

GPU Smith's engineering scope includes infrastructure integration, networking, power, cooling and the practical operation of inference workloads. Our published articles explain the tradeoffs behind deployment, procurement and ongoing operations so teams can ask better questions and document decisions.

Work with GPU Smith

Explore the [hardware library](#), [GPU references](#), [networking references](#) and [vendor research](#). [Contact GPU Smith](#) with your workloads and deployment constraints to discuss a project and confirm current scope, pricing and availability.

Inclusion of a manufacturer or product in our research does not imply a partnership, certification or endorsement.

Website: <https://gpusmith.com>

This article is educational. Verify consequential medical, legal, financial, regulatory and technical decisions with appropriate professionals and the cited primary sources. Company services are described separately from the article's research findings.