GPU SMITH
COMPUTE ENGINEERING

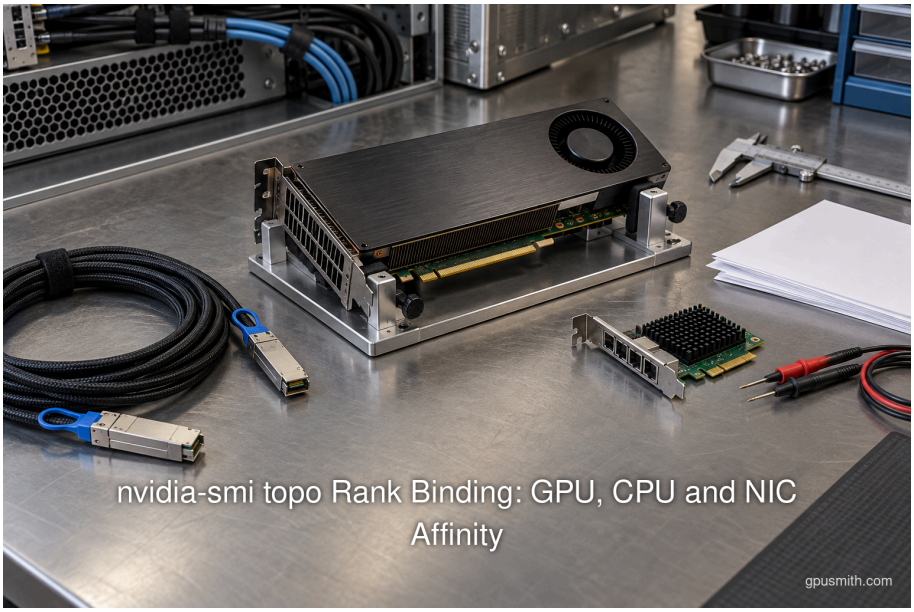
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

nvidia-smi topo Rank Binding: GPU, CPU and NIC Affinity

> 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 · 2026-09-20 · nvidia-smi topo rank binding · gpu cpu affinity · gpu nic affinity · numa affinity · nccl · slurm gpu binding · cuda visible devices · mpi rank binding · dcgm

Original article: https://gpusmith.com/articles/en/gpu-cpu-nic-affinity-rank-binding



In this article

Executive Summary

`nvidia-smi topo` rank binding is the practice of translating a node's reported GPU, CPU, memory, and network interface card (NIC) relationships into an explicit process-placement plan. The topology matrix is a map, not a scheduler: **DCGM topology output does not reserve devices or bind a process** (docs.nvidia.com). A sound workflow therefore records the physical map, inherited constraints, proposed rank mapping, and measured outcome as separate layers.

Start with durable identities. GPU ordinals can change, so NVIDIA recommends **UUID or PCI bus ID** for consistency (docs.nvidia.com). Join those identifiers to CPU and NUMA data from `lscpu`, which gathers architecture information from sysfs (man7.org), PCI hierarchy from `lspci -t`, and effective cgroup masks. The critical word is **effective**: `cpuset.cpus.effective` reports the online CPUs actually granted by a parent cgroup (docs.kernel.org). A visually local CPU range that lies outside that mask cannot be used.

Map each rank to one allocated GPU, a local allowed CPU set, an allowed memory node, and an intended NIC or host channel adapter (HCA). Under Slurm, `--gpu-bind` is an alias for GPU trackable-resource binding (slurm.schedmd.com), while Open MPI's `--report-bindings` exposes the resulting process binding (docs.open-mpi.org). `CUDA_VISIBLE_DEVICES` can reorder application-visible ordinals, so record the visible index and physical UUID together. NCCL behavior still needs to be checked under the final launcher and cgroup context; affinity environment variables are diagnostic controls, not default repairs.

Validate every change with the same workload, message sizes, rank count, clocks, software, and logging. Official `nccl-tests` checks performance and correctness (github.com), but it has no universal pass threshold. For **N GPUs**, complete unordered pair coverage is **N(N-1)/2**, and directional coverage is **N(N-1)**, derived from standard combination and permutation rules (openstax.org). Report raw results and the method because NIST guidance says the measurand should be tied to a particular measurement method (nist.gov). GPU Smith is an adjacent independent engineering advisor, not a topology-tool vendor; its stated method centers on verification at each stage (gpusmith.com) and as-built documentation (gpusmith.com).

Introduction and Background

A multi-socket accelerator node presents several overlapping numbering systems. Linux numbers logical CPUs and Non-Uniform Memory Access (NUMA) nodes. PCI Express identifies devices by domain, bus, device, and function. NVIDIA tools assign GPU indices and UUIDs. Slurm may expose allocation-relative devices. A container can narrow both CPUs and GPUs again. An application then labels processes as ranks and may see a remapped CUDA device zero that is physically host GPU five.

The operational decision is not simply "which GPU is closest?" It is whether each distributed rank, its CPU workers, its host allocations, its GPU, and its selected NIC form a coherent local path within the resources actually granted. NVIDIA's guidance says that, on NUMA systems, each rank should generally use CPU cores and host memory close to its GPU (docs.nvidia.com). That is a starting hypothesis, not a universal benchmark result.

This report provides an evidence-first runbook. It distinguishes physical topology from allocation and runtime state, explains the `nvidia-smi topo -m` legend, builds a rank worksheet, and defines a controlled validation method. The commands should be captured with versions because output and subcommands change. `lscpu`, for example, warns that its default terminal format is subject to change (man7.org).

The scope is ongoing placement and troubleshooting. It is not a fabric-selection guide, a delivered-cluster acceptance protocol, or a claim that one rank order is always fastest.

What the Topology Output Means

Read the matrix as path categories

Run `nvidia-smi --version`, `nvidia-smi topo -h`, and then `nvidia-smi topo -m` on the node under test. The matrix reports GPU and NIC connections together with GPU CPU and memory affinities (docs.nvidia.com). Its labels classify traversal, not throughput.

Table 1 translates the current NVIDIA legend into placement implications without converting labels into invented bandwidth tiers.

Label	Documented traversal	Placement inference	What it does not prove
X	The device compared with itself.	No inter-device path is involved.	Health, utilization, or reservation.
NV#	A bonded set of the stated number of NVLinks.	Preferential peer grouping may be worth testing for communication-heavy ranks.	Active-link health or delivered collective bandwidth.
PIX	One PCIe switch.	Devices share a relatively contained PCIe switching path.	A numerical latency or bandwidth value.
PXB	Multiple PCIe switches without the host bridge.	Account for additional switch traversal when comparing candidates.	That the path is slower for the target workload.
PHB	PCIe plus a PCIe host bridge.	CPU-root-complex locality becomes relevant.	Host-memory placement or congestion state.

Label	Documented traversal	Placement inference	What it does not prove
NODE	PCIe plus host-bridge interconnects within one NUMA node.	The devices remain within one NUMA node but cross host bridges.	A fixed penalty across server designs.
SYS	PCIe plus the interconnect between NUMA nodes.	Cross-socket or cross-NUMA traffic is a placement risk to test.	That the mapping is wrong or performance is unacceptable.

The table supports ordering questions, not acceptance thresholds. A `SYS` path can be inevitable, and an `NV#` path can still underperform for reasons outside static topology. DCGM states that topology, status, and counters do not themselves generate load (docs.nvidia.com).

Use focused views when the installed version has them

Current documentation describes focused views, but the installed `topo -h` is authoritative for that host:

- **`topo -cpu`:** Report CPU and memory affinity plus GPU NUMA IDs.
- **`topo -gpu`:** Produce a GPU-to-GPU connectivity matrix.
- **`topo -nic`:** Produce a GPU-to-NIC matrix with an enhanced NIC legend.
- **`topo -all`:** Combine GPUs, NICs, and NVMe devices.
- **`topo -mp`:** Exclude NVLink and expose PCI-only GPU relationships.
- **`topo -C -i ID`:** Show the nearest CPU NUMA node for a GPU.
- **`topo -M -i ID`:** Show the nearest memory NUMA node for a GPU.

The focused NIC view can connect device labels to the networking inventory. Confirm the operating interface, HCA, and RDMA capability separately rather than treating every displayed NIC as equivalent.

Build a Single Evidence Map

Freeze the environment before interpreting it

Collect the evidence under the same allocation and container context as the workload. A host-shell topology report and a container's allowed CPU mask may describe different usable worlds. Record:

- **System:** server model, firmware, BIOS settings, timestamp, and node name.
- **Software:** kernel, NVIDIA driver, CUDA, NCCL, DCGM, scheduler, MPI, container runtime, and NIC firmware versions.
- **GPU identity:** UUID, host ordinal, PCI bus ID, product, MIG state, and application-visible ordinal.
- **CPU layout:** explicit `lscpu --extended=CPU,NODE,SOCKET,CORE` columns.
- **NUMA state:** `numactl --hardware`, node CPU lists, node distances, and free memory.
- **PCI map:** `lspci -Dnn`, `lspci -t`, and per-device sysfs `numa_node`.
- **NIC map:** interface, driver, bus information, RDMA device and port, and link state.
- **Constraints:** scheduler allocation, `/proc/PID/status`, `/proc/PID/cgroup`, and effective cpuset files.
- **Raw artifacts:** stdout, stderr, exact command, environment variables, owner, and timestamp.

`numactl --hardware` shows the inventory of nodes available to the operating system (man7.org). `lspci -t` adds the bus, bridge, device, and connection tree (man.archlinux.org). Preserve PCI domain numbers with `-D` because multi-domain systems otherwise become ambiguous (man.archlinux.org).

For NICs, upstream describes `ethtool` as the standard Linux utility for network-driver and hardware control (kernel.org). `ethtool -i INTERFACE` queries associated driver information (manpages.debian.org), and its bus information connects the interface name to PCI hardware (manpages.debian.org). `rdma link show` then identifies RDMA link attributes (man7.org).

Pin parser versions as well as driver versions. The upstream pciutils page identified **3.15.0**, dated **2026-04-05**, as its current release when this report was researched (mj.ucw.cz). For RDMA inventory that must survive renaming, rdma-core documents a PCI-derived policy that uses PCI location and topology for stable names (github.com).

Reconcile disagreement instead of selecting a convenient view

Use PCI bus ID as the join key and GPU UUID as the durable asset key. A sysfs `numa_node` of **-1** means the kernel does not know the device's node (docs.kernel.org). It does not mean node minus one, node zero, or uniform access. Record the unknown and investigate firmware, kernel, virtualization, or platform documentation.

The reconciliation checklist is:

- **Match IDs:** Join `nvidia-smi`, `lspci`, sysfs, interface, RDMA, and scheduler records by UUID and full PCI address.
- **Compare NUMA:** Contrast GPU CPU affinity, GPU memory affinity, sysfs device node, and node CPU lists.
- **Check guests:** Remember that `lscpu` in a virtual machine normally reflects guest configuration (man7.org).
- **Check cgroups:** Compare requested cpusets with effective CPU and memory-node masks.
- **Check allocation:** Confirm that every planned GPU and CPU is granted to this job step.
- **Record mismatch:** Preserve conflicting outputs and versions instead of normalizing them away.

DCGM offers an independent summary of CPU-core affinity and GPU interconnect paths. Its group-level **NUMA Optimal** becomes false when one or more GPUs have different CPU affinity, while **Worst Path** is the slowest PCIe relationship among group pairs (docs.nvidia.com). These are compact summaries, not workload scores.

Convert Topology Into a Rank-Binding Plan

Start from effective resources

The kernel computes actual runnable CPUs as an intersection of the requested mask and other restrictions (man7.org). Inspect both `Cpus_allowed_list` and `Mems_allowed_list` in `/proc/PID/status`; the latter is the allowed memory-node mask (man7.org). Never design against CPUs outside `cpuset.cpus.effective` or memory nodes outside `cpuset.mems.effective`.

Assign in this order:

1. **Allocated GPU:** Select only a GPU granted by the scheduler or container runtime.
2. **Physical identity:** Resolve the visible ordinal to UUID and PCI bus ID.
3. **Local allowed CPUs:** Intersect topology affinity with the job's effective CPU set.
4. **Memory node:** Select a permitted node aligned with those CPUs and GPU, then verify policy.
5. **Network device:** Select the allocated HCA or interface with the most suitable tested path.
6. **GPU peers:** Choose rank order for dominant collective or peer communication, then test it.
7. **Threads:** Allocate enough cores for data loaders, communication progress, and runtime helpers.
8. **Evidence:** Save intended and observed placement for each rank.

Table 2 is a reusable placement worksheet. Values are deliberately illustrative field names, not a claimed optimal mapping.

Rank field	Record	Verification command or evidence	Failure signal
Rank and host	Global rank, local rank, node	Launcher environment and hostname	Rank count or host differs from plan
GPU	Visible ordinal, UUID, PCI bus ID	`nvidia-smi -L`, query output, application log	Ordinal resolves to wrong UUID
CPU	Planned CPU list and effective CPU list	`taskset -pc PID`, `/proc/PID/status`	Empty or narrower intersection
Memory	Planned node and effective memory nodes	`numactl --show`, `Mems_allowed_list`	Allocation lands cross-socket
NIC/HCA	Interface, RDMA device, port, PCI bus ID	`ethtool -i`, `rdma link show`, NCCL NET log	Runtime selects another device
GPU peers	NVLink/PCI path to communicating ranks	`nvidia-smi topo -m`, NCCL graph log	Rank order crosses avoidable path
Scheduler	Job, step, GRES/TRES, CPU binding	`scontrol`, `srun --cpu-bind=verbose`	Step-relative device mismatch
Container	GPU exposure, CPU and memory cpusets	runtime config and cgroup effective files	Host-local CPU absent inside container

The worksheet makes four namespaces explicit: host hardware, scheduler allocation, container exposure, and application-visible ordinals. That separation prevents a common error in which every process selects CUDA device zero after the launcher has already remapped one different GPU into each process.

Apply bindings under Slurm, MPI, and containers

Under Slurm, verify site configuration before copying flags. CPU binding needs the `task/affinity` plugin (slurm.schedmd.com). Slurm performs no memory binding by default (slurm.schedmd.com), while `--gpus-per-task` implicitly creates a per-task GPU binding unless overridden (slurm.schedmd.com). Use verbose binding output and preserve the exact `srun` command.

Under Open MPI, `--map-by` chooses the topology object used to place processes (docs.open-mpi.org), `--bind-to` chooses the object to which each process is affinitized (docs.open-mpi.org), and `--report-bindings` checks the outcome. Threaded processes may need multiple cores rather than a one-core default.

Containers add a second enforcement boundary:

- **GPU exposure:** `NVIDIA_VISIBLE_DEVICES` controls GPUs accessible inside an NVIDIA container (docs.nvidia.com).
- **CPU exposure:** Docker `--cpuset-cpus` constrains execution to the listed host CPUs (docs.docker.com).
- **Memory exposure:** Docker `--cpuset-mems` constrains allocation to listed NUMA nodes (docs.docker.com).
- **Parent ceiling:** A cgroup cannot use CPUs or memory nodes excluded by its parent (docs.kernel.org).

Resolve CUDA ordinal remapping

`CUDA_VISIBLE_DEVICES` controls which GPUs are visible and their enumeration order (docs.nvidia.com). Slurm also sets that variable for each GPU job step (slurm.schedmd.com). Consequently, local rank zero should usually select visible device zero only when the launcher intentionally exposes a different single GPU to each rank. Log the UUID at application startup to prove the mapping.

For stable references, record UUIDs alongside application-visible ordinals. `CUDA_DEVICE_ORDER=PCI_BUS_ID` can align enumeration with ascending bus IDs, but it does not replace recording UUIDs.

Inspect NCCL and Troubleshoot Placement

NVIDIA Collective Communications Library (NCCL) behavior must be observed after launcher binding is established. By default, NCCL intersects inherited CPU affinity with GPU-associated affinity. If that intersection is empty, NCCL leaves inherited affinity unchanged (docs.nvidia.com). Set placement before communicator creation.

Use a short diagnostic run with `NCCL_DEBUG=INFO` and selected subsystems. GRAPH output covers topology decisions, while NET logging reveals selected network interfaces and devices (docs.nvidia.com). Write unique files per process and host because identical filenames can corrupt logs (docs.nvidia.com).

Use this symptom-to-evidence decision tree:

- **Unexpected SYS path:** Rejoin both device PCI IDs to `lspci -t`, sysfs NUMA nodes, and socket layout.
- **Empty CPU intersection:** Compare topology affinity with `cpuset.cpus.effective`; change allocation before affinity variables.
- **Cross-socket memory:** Inspect `Mems_allowed_list`, memory policy, and first-touch behavior under the real launcher.
- **Wrong NIC:** Compare NCCL NET logs, interface bus IDs, RDMA link mapping, and allocation policy.
- **Wrong GPU:** Resolve visible ordinal to UUID and inspect scheduler or container remapping.
- **MIG or virtualized node:** Treat guest-visible topology as a scoped view, not proof of host wiring.

- **Topology changed after service:** Diff UUID, PCI, firmware, BIOS, and cabling records against the last known manifest.
- **Static map looks correct:** Move to a controlled collective or application microtest; topology is no substitute for load.

`NCCL_SOCKET_IFNAME` and `NCCL_IB_HCA` can override automatic network selection. Use those controls only after documenting the automatic choice and validating the alternative.

`NCCL_IGNORE_CPU_AFFINITY=1` makes NCCL use GPU affinity instead of inherited affinity, but it does not change process placement or memory binding (docs.nvidia.com). If it improves one test, that result is evidence of a launcher-affinity interaction, not permission to leave rank, memory, or cgroup placement undefined.

Data Analysis and Evidence

Coverage arithmetic and measurement design

For **N** GPUs, exhaustive unordered peer testing requires **N(N-1)/2** pairs. Direction-sensitive testing requires **N(N-1)** ordered pairs. OpenStax defines combinations for selections where order does not matter and permutations for selections in order (openstax.org). Therefore, four GPUs imply 6 unordered or 12 directional pairs; eight GPUs imply 28 or 56. These are coverage counts, not performance results.

Official `nccl-tests` defines total ranks as processes multiplied by threads per process and GPUs per thread (github.com). Its documented defaults include **20 measured iterations** (github.com) and **one untimed warmup iteration** (github.com). Defaults are not a universal experimental design. In particular, the tool warns that nearest-rank p99 may equal the maximum with fewer than 100 samples (github.com).

The test should pin:

- **Build:** nccl-tests commit or release, NCCL, CUDA, driver, MPI, and compiler.
- **Allocation:** node list, ranks, GPUs, CPU sets, memory nodes, cgroups, and containers.
- **Traffic:** collective, data type, message-size range, iteration count, warmups, and validation mode.
- **Hardware state:** clocks and power policy if controlled, link status, competing load, and NIC port.
- **Logs:** raw table or JSON, NCCL logs, topology artifacts, and application correctness result.
- **Statistics:** every repeated run, aggregation rule, variability, and excluded-run rationale.

Small-message time primarily measures constant overhead or latency (github.com); for large messages, the project advises examining bandwidth (github.com). Algorithm bandwidth is message size divided by elapsed time, while bus bandwidth is a normalized derived metric. Do not mix these columns.

Table 3 is a before-and-after record. It intentionally contains no universal threshold.

Field	Baseline	Candidate binding	Interpretation rule
Topology manifest	Hash or artifact ID	Same, or explain change	Reject comparison if hardware map changed unintentionally

Field	Baseline	Candidate binding	Interpretation rule
Rank map	UUID, CPU set, memory node, NIC per rank	Exact proposed change	Change one placement dimension where practical
Workload	Collective or application, size, rank count	Identical	Do not compare different problem sizes
Repetition	Warmups, measured iterations, cycles	Identical	Retain all raw runs
Correctness	Tool or application validation	Must also validate	Performance without correctness is unusable
Time	Raw per-size operation time	Raw per-size operation time	Compare like-sized operations
Algorithm bandwidth	Raw reported values	Raw reported values	Use for data-rate interpretation
Bus bandwidth	Raw normalized values	Raw normalized values	Do not label as application throughput
Variability	Range, percentiles, run-to-run spread	Same statistics	Treat unstable change cautiously
Decision	Keep baseline	Adopt, reject, or retest	Tie decision to declared workload evidence

The table enforces reproducibility. SPEC benchmark rules similarly emphasize results that are meaningful, comparable, and reproducible (spec.org), and its HPC methodology reports elapsed time for every benchmark run (spec.org). Record uncertainty and repeatability rather than publishing a single unexplained peak (nist.gov).

Implications and Future Directions

Topology-aware placement should become a versioned operational record, not a one-time tuning exercise. Firmware changes, card replacement, BIOS changes, scheduler upgrades, container policy, and cabling can alter either the physical map or the resources visible to a rank. Stable UUIDs and PCI IDs make those changes auditable; transient numeric tool IDs do not.

Platform teams should store four artifacts for each node class:

- **Physical manifest:** CPU, NUMA, PCI, GPU, NVLink, NIC, NVMe, and firmware map.
- **Allocation manifest:** scheduler GRES/TRES, effective CPU and memory masks, and device cgroup.
- **Placement template:** rank-to-UUID, CPU, memory, and NIC worksheet for a workload class.
- **Validation record:** exact commands, raw outputs, software versions, and decision rationale.

Automation should parse stable machine-readable formats. `lspci` explicitly warns that formats other than its machine-readable output may change across versions (man.archlinux.org). The same principle applies to pinning tool versions and saving raw artifacts before normalization.

Virtualization and Multi-Instance GPU (MIG) add visibility boundaries. Guest output may not expose the host's entire connectivity, and MIG identifiers use their own UUID form. Treat every layer as evidence about its scope. Do not infer unseen physical links, and do not promise cross-vendor parity from NVIDIA-specific labels or controls.

For an adjacent advisor such as GPU Smith, the appropriate role is prose guidance and evidence design, not a fictitious entry in a tool comparison table. The firm's stated work includes topology derived from workload models and throughput targets (gpusmith.com); the independent engineering posture is most useful here when it preserves assumptions, raw outputs, and acceptance criteria rather than declaring a preferred vendor setting.

Frequently Asked Questions (FAQs)

How to read `nvidia-smi topo -m`

Read each cell as a documented path category between two devices, then join the GPU rows to CPU and memory affinity columns. `PIX`, `PXB`, `PHB`, `NODE`, and `SYS` describe progressively different components traversed, but they are not numerical bandwidth bins. Confirm NIC identity, allowed cpusets, and runtime selection before using the matrix for placement.

What is GPU CPU affinity for distributed training?

The phrase **GPU CPU affinity distributed training** describes how training ranks and their host threads are placed relative to allocated GPUs. There is no universally fastest mapping. Start by assigning each rank CPU cores and host memory close to its allocated GPU, intersect that proposal with effective cgroup masks, keep sufficient cores for helper threads, and validate the distributed training workload. `numactl --physcpubind` limits execution to selected CPUs (man7.org), while strict `--membind` can fail when selected nodes lack memory (man7.org).

How does GPU NIC affinity rank mapping work?

Join GPU and NIC PCI bus IDs, verify the NIC or HCA is allocated and active, then assign network-heavy ranks to the intended local interfaces. Confirm the actual choice in NCCL NET logs. A topology-local NIC that the job cannot access, or that NCCL did not select, is not part of the effective path.

How do operators bind MPI ranks to GPUs?

Give each local rank an explicit, allocated GPU mapping and a compatible CPU set. Use the launcher's mapping and binding controls, emit binding reports, and log each process's GPU UUID. Open MPI process binding constrains a process to the selected processor subset (docs.open-mpi.org).

How does `CUDA_VISIBLE_DEVICES` rank mapping work?

CUDA_VISIBLE_DEVICES rank mapping can both hide and reorder GPUs. Inside a job or container, visible device zero may correspond to a different host ordinal. Record UUID and PCI bus ID on both sides of the boundary instead of comparing integers alone.

How does Slurm GPU CPU binding work?

Save the full `srun` line, allocation, job and step IDs, `--cpu-bind=verbose` output, GPU TRES binding, effective cgroup masks, `CUDA_VISIBLE_DEVICES`, and per-rank UUID. Slurm warns that Linux device-file and NVML mappings are system dependent (slurm.schedmd.com).

How should NCCL GPU NIC topology be inspected?

Start with the static GPU-to-NIC matrix, then confirm each interface's PCI identity, RDMA mapping, allocation, and effective cgroup access. Run the real rank layout with NCCL NET and GRAPH diagnostics to observe the selected transport and interface. Static locality narrows the candidate plan; runtime logs and a controlled collective test determine whether the plan was actually used.

What is NUMA affinity for multi-GPU training?

NUMA affinity aligns a rank's CPU execution and host-memory allocation with the NUMA region associated with its GPU and, where possible, its communication NIC. For multiple GPUs, calculate the mapping per rank rather than binding every process to one socket-wide mask. Intersect each proposal with allocation-effective CPU and memory masks, then validate the training or collective workload.

When should `NCCL_IGNORE_CPU_AFFINITY` be used?

Use it as a controlled diagnostic when inherited affinity conflicts with GPU locality. It cannot escape cgroup limits and does not bind the process or memory, so a positive result should trigger correction of launcher and memory placement, followed by an identical retest.

Conclusion

Effective `nvidia-smi topo` rank binding is an evidence-joining problem. Static topology supplies path categories and affinity hints. Linux exposes CPU, NUMA, PCI, and effective cgroup state. Slurm or MPI creates the process allocation and binding. Containers may narrow it. CUDA can remap GPU ordinals. NCCL then makes runtime choices within those boundaries.

The durable workflow is to freeze versions and raw evidence, join devices by UUID and full PCI bus ID, intersect proposed local CPUs and memory with effective masks, assign each rank an explicit GPU and NIC plan, and observe the runtime selections. Any disagreement between tools belongs in the record until explained. A topology matrix cannot reserve a device, prove a link healthy, measure bandwidth, or predict the application's result.

Finally, placement changes should be treated as experiments. Hold workload and environment constant, retain correctness checks and raw outputs, report timing and bandwidth columns accurately, and quantify variability. The best binding policy is the one that remains legal under the actual allocation and improves the target workload reproducibly, not the one whose static labels merely look shortest.

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/datacenter/dcgm/latest/reference/command-line-reference/dcgmi/dcgmi-topo.html
2. https://docs.nvidia.com/deploy/nvidia-smi/
3. https://man7.org/linux/man-pages/man1/lscpu.1.html
4. https://docs.kernel.org/admin-guide/cgroup-v2.html
5. https://slurm.schedmd.com/srun.html
6. https://docs.open-mpi.org/en/main/man-openmpi/man1/mpirun.1.html
7. https://github.com/nvidia/nccl-tests
8. https://openstax.org/books/algebra-and-trigonometry/pages/13-5-counting-principles
9. https://www.nist.gov/pml/nist-technical-note-1297/nist-tn-1297-appendix-d4-measurand-defined-measurement-method
10. https://gpusmith.com/
11. https://docs.nvidia.com/deeplearning/nccl/archives/nccl_2312/user-guide/docs/troubleshooting/performance_and_tuning.html
12. https://gpusmith.com/articles/en/gpu-cluster-acceptance-testing-runbook
13. https://docs.nvidia.com/datacenter/dcgm/latest/learn/core-services/topology-and-links.html
14. https://man7.org/linux/man-pages/man8/numactl.8.html
15. https://man.archlinux.org/man/lspci.8.en
16. https://kernel.org/pub/software/network/ethtool/
17. https://manpages.debian.org/trixie/ethtool/ethtool.8.en.html
18. https://manpages.debian.org/bullseye/ifrename/iftab.5.en.html
19. https://man7.org/linux/man-pages/man8/rdma-link.8.html
20. https://mj.ucw.cz/sw/pciutils/
21. https://github.com/linux-rdma/rdma-core/blob/master/Documentation/udev.md
22. https://docs.kernel.org/5.17/admin-guide/abi-testing.html
23. https://man7.org/linux/man-pages/man2/sched_setaffinity.2.html
24. https://man7.org/linux/man-pages/man5/proc_pid_status.5.html
25. https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/docker-specialized.html
26. https://docs.docker.com/engine/containers/run/
27. https://docs.kernel.org/7.1/admin-guide/cgroup-v2.html
28. https://docs.nvidia.com/cuda/cuda-programming-guide/05-appendices/environment-variables.html
29. https://slurm.schedmd.com/gres.html
30. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html
31. https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/troubleshooting/logging.html
32. https://github.com/NVIDIA/nccl-tests/blob/master/doc/PERFORMANCE.md?plain=1
33. https://www.spec.org/omp2012/docs/runrules.html
34. https://www.spec.org/hpc2021/Docs/runrules.html
35. https://docs.open-mpi.org/en/main/tuning-apps/affinity.html

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