GPU SMITH
COMPUTE ENGINEERING

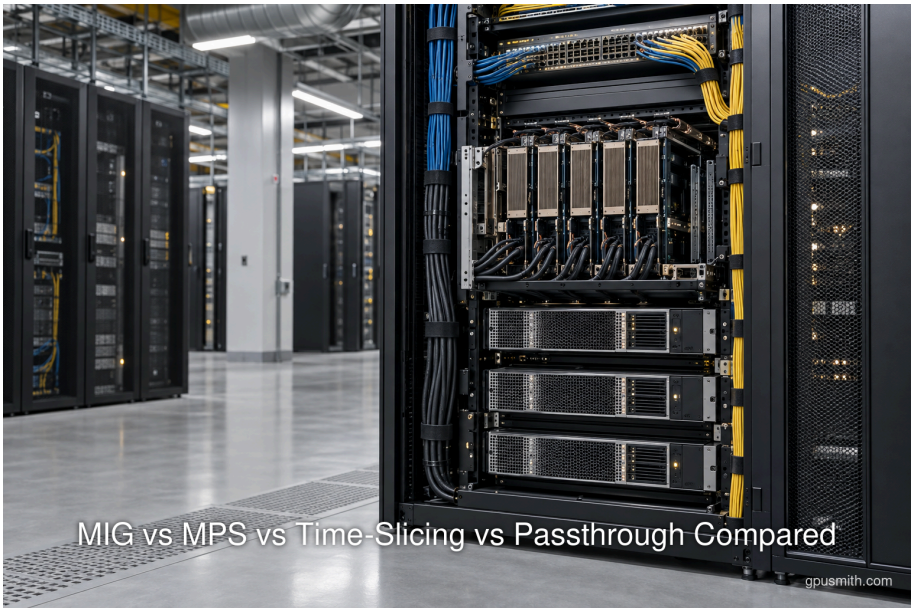
GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

MIG vs MPS vs Time-Slicing vs Passthrough Compared

> 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 · mig vs mps · gpu time-slicing · gpu passthrough · gpu sharing · nvidia mig · cuda mps · gpu operator · gpu virtualization · kubernetes gpu

Original article: https://gpusmith.com/articles/en/mig-vs-mps-time-slicing-gpu-passthrough



In this article

Executive Summary

The choice among **Multi-Instance GPU (MIG)**, **CUDA Multi-Process Service (MPS)**, **GPU Operator time-slicing**, **full-device passthrough**, and a **dedicated GPU** starts with the trust boundary, not utilization. MIG partitions supported NVIDIA GPUs into isolated instances with dedicated compute and memory resources (docs.nvidia.com). Time-slicing, by contrast, has no memory or fault isolation between replicas (docs.redhat.com). MPS gives cooperating processes concurrent kernel execution and separate address spaces, but its active-thread percentage is a ceiling, not reserved hardware (docs.nvidia.com). Passthrough assigns an entire physical device to one guest and does not share it (docs.openstack.org).

The short decision is: use **MIG** when supported hardware must serve separate workloads with hardware-backed memory and fault boundaries; use **MPS** when trusted processes form one cooperative CUDA application; use **time-slicing** for bursty, interactive, mutually trusted jobs that can tolerate contention; and use **passthrough** when a virtual machine needs the whole device. Choose **dedicated bare metal** when strict latency, whole-device reset control, unsupported MIG hardware, maximum topology control, or a simple single-owner operating model matters more than consolidation. AWS explicitly recommends evaluating bare metal for performance-sensitive workloads with strict latency requirements (docs.aws.amazon.com).

Quantities exposed to a scheduler require careful interpretation. Kubernetes extended resources are whole numbers and cannot be overcommitted (kubernetes.io), yet a time-slicing configuration can make one physical GPU advertise several integer replicas. A documented GKE example combines seven MIG partitions with three clients each to expose up to **21 shared devices** (docs.cloud.google.com), but requesting two replicas does not guarantee twice the compute. GKE says a shared-GPU resource count is not a measure of compute power (docs.cloud.google.com).

No mechanism makes performance deterministic by name alone. A 2026 study reported favorable MPS cases with up to **30%** better performance and about **20%** lower energy, but also about **30%** worse performance under memory contention (arxiv.org) (arxiv.org). Therefore the procurement decision should be conditional on measured p50 and p95 latency, throughput, out-of-memory behavior, cross-workload fault behavior, utilization, accounting quality, and reset effects on the actual target stack.

Introduction and Background

GPU consolidation is often presented as a capacity question: how many jobs can fit on one accelerator? For private-AI platform owners, Kubernetes and Slurm operators, security architects, and infrastructure buyers, that framing is incomplete. The decisive questions are whether workloads trust one another, whether memory must be isolated, whether one fault may affect another workload, and whether performance objectives survive contention. **GPU sharing is a set of different resource-control models, not one continuum of smaller GPU portions.**

This report compares NVIDIA MIG vs MPS, MIG vs time slicing, GPU time slicing vs passthrough, and MPS vs time slicing GPU behavior. It also answers the broader searches “GPU sharing methods comparison,” “NVIDIA GPU sharing options,” “MIG vs GPU passthrough,” and “best GPU virtualization method for AI workloads.” The scope is allocation and isolation semantics as verified on **September 20, 2026**. It excludes a Kubernetes Dynamic Resource Allocation migration runbook and fractional-L4 economics.

Kubernetes has stable GPU scheduling support since version **1.26**, using vendor device plugins to expose custom schedulable resources (kubernetes.io) (kubernetes.io). Slurm uses Generic Resources, or GRES, and does not allocate one unless the job requests it (slurm.schedmd.com). Those scheduler objects say what can be placed. They do not, without an underlying mechanism, prove a memory boundary, fixed compute share, or fault-containment boundary.

GPU Smith is an adjacent independent engineering adviser, not one of the sharing mechanisms. Its relevant perspective is to specify acceptance criteria before procurement: its published method says engagements are delivered against written acceptance criteria (gpusmith.com). That principle is applied here as a neutral testable decision process, not as evidence for any NVIDIA feature.

Constraint-First Decision Tree

The following yes/no flowchart deliberately starts with failure consequences. A “yes” sends the reader to the indented next test; a “no” sends the reader to the named alternative.

1. **Must different trust domains share one physical GPU?**
    - **Yes:** require a documented isolation boundary. Continue to question 2.
    - **No:** if processes cooperate as one application, evaluate MPS. If jobs are unrelated but mutually trusted and bursty, evaluate time-slicing.
2. **Must GPU memory and faults be separated in hardware?**
    - **Yes:** use MIG on a supported GPU, or put each tenant on a separate passed-through or dedicated GPU.

    - **No:** time-slicing may be acceptable if contention and shared failure impact pass testing.
3. **Does each workload require a separate virtual machine and guest operating system?**
    - **Yes:** choose full-device passthrough for one VM per GPU, or separately evaluate licensed vGPU products outside this comparison.
    - **No:** continue to question 4.
4. **Can the model and its working set fit a supported MIG profile?**
    - **Yes:** test MIG with the exact profile geometry.
    - **No:** use passthrough or a dedicated GPU. Do not make time-slicing a memory-capacity substitute because shared jobs still see and contend for the same physical memory pool.
5. **Are the processes coordinated, trusted, and designed to run concurrently?**
    - **Yes:** MPS is a candidate, especially when individual processes underfill the GPU.
    - **No:** continue to question 6.
6. **Are workloads bursty, interactive, and tolerant of variable service time?**
    - **Yes:** time-slicing is the lowest-friction consolidation candidate. Google explicitly recommends it for bursty and interactive workloads with idle periods (docs.cloud.google.com).
    - **No:** use MIG, passthrough, or a dedicated GPU, then validate latency under load.
7. **Is strict latency, full reset control, or maximum topology access non-negotiable?**
    - **Yes:** the correct result may be **none of these sharing modes**, meaning one dedicated accelerator per workload.
    - **No:** select the least complex mechanism that passes the worksheet later in this report.

The flowchart is a filter, not a proof of fitness. A candidate that passes its branch still needs driver, GPU model, container runtime, scheduler, monitoring, and workload-level validation.

Multi-Instance GPU

Capabilities

MIG is spatial partitioning. A supported GPU is divided into GPU instances with dedicated compute and memory. NVIDIA states that memory traffic has separate and isolated paths through the entire memory system (docs.nvidia.com). This is the strongest sharing option in the comparison when several containerized or process workloads need hardware-backed separation while remaining on one card.

MIG begins with the NVIDIA **Ampere** generation (docs.nvidia.com). Support is model-specific, and profile geometry is not interchangeable across all GPUs. GKE's current supported list includes GB200, B200, H200, H100, A100, and RTX PRO 6000 (docs.cloud.google.com). The operator must verify the exact board, driver branch, CUDA version, and permitted profiles rather than treating "Ampere or newer" as a complete compatibility test.

Adoption and Scheduler Fit

Slurm has supported MIG devices since **21.08** and can apply cgroup isolation and task binding (slurm.schedmd.com) (slurm.schedmd.com). Its important operational limitation is that MIG devices must already be partitioned; Slurm does not dynamically create the geometry (slurm.schedmd.com). An AWS EKS

implementation used node labels to define specific partition strategies (aws.amazon.com).

Strengths and Limitations

- **Best fit:** Separate services or teams that need memory and fault containment on one supported GPU.
- **Capacity:** A supported GPU can expose up to **seven** GPU instances, depending on model and profile geometry (docs.nvidia.com).
- **Reconfiguration:** Instance creation and deletion can cause placement fragmentation, so desired geometry should be treated as node state, not an invisible scheduler detail.
- **Persistence:** Created MIG devices do not survive a system reboot and require configuration reconciliation.
- **Monitoring:** Metric availability and attribution differ by GPU generation, orchestrator, and metric, so the target stack needs an explicit telemetry test.
- **Accounting caveat:** Slurm does not provide gpumem or gpuutil accounting for MIG devices (slurm.schedmd.com).

CUDA Multi-Process Service

Capabilities

MPS is a process-concurrency service, not a hardware partition. Its central use case is cooperative processes acting as one application (docs.cloud.google.com). It allows multiple processes to share one GPU context (aws.amazon.com).

Each client owns an address space, but MPS offers limited error containment (docs.nvidia.com). A fatal GPU fault can be reported to every client using the affected GPU subset. This makes MPS appropriate for processes within one administrative and application trust domain, not as a substitute for MIG between mutually untrusted tenants.

Adoption and Scheduler Fit

Google characterizes MPS as optimal for cooperative processes acting as a single application (docs.cloud.google.com). Slurm models MPS allocations as percentages and translates the configured resource count into a percentage (slurm.schedmd.com). That percentage still requires careful interpretation because an active-thread limit constrains use rather than reserving dedicated streaming multiprocessors.

As of September 2026, the optional MPS v3 memory-partitioning feature requires Linux cgroup v2 and **CUDA 13.4 or newer**, and that feature explicitly does not support MIG devices (docs.nvidia.com) (docs.nvidia.com).

Strengths and Limitations

- **Best fit:** Trusted MPI ranks, inference workers, or cooperative services that underfill one GPU individually.
- **Concurrency:** Different processes can overlap work without being rewritten into one process.
- **Client scale:** The documented default supports up to **60 client CUDA contexts per device** (docs.nvidia.com).

- **Platforms:** MPS is supported on Linux and QNX, not as a general Windows host mechanism.
- **Isolation:** Address-space separation is not equivalent to hardware fault or performance isolation.
- **Accounting:** Standard tools may attribute client activity to the MPS server process, complicating per-job chargeback.

GPU Operator Time-Slicing

Capabilities

Time-slicing is temporal sharing through advertised replicas. The NVIDIA device plugin lets operators define replicas that Kubernetes can hand independently to pods. Workloads then interleave on an oversubscribed physical GPU. The crucial boundary is explicit: replicas have neither memory nor fault isolation (docs.redhat.com). A pod can run an unlimited number of processes on its assigned time-sliced GPU resource.

Kubernetes itself still sees integer extended resources. If both a GPU request and limit are present, they must be equal (kubernetes.io). This integer accounting does not mean each replica is a fixed fraction. Google states directly that a time-sharing or MPS resource count does not measure compute power assigned to a container (docs.cloud.google.com).

Adoption and Scheduler Fit

Time-slicing fits Kubernetes clusters with many low-duty-cycle notebooks, development pods, or light inference endpoints. AWS published an EKS example that scaled a TensorFlow deployment to **20 replicas** (aws.amazon.com). That is an implementation example, not evidence that every workload can sustain 20-way sharing.

GKE caps time-sharing at **48 containers per physical GPU** and documents a combined example with seven MIG partitions and three shared clients per partition, exposing up to **21** time-sharing devices (docs.cloud.google.com (docs.cloud.google.com). These are platform limits, not recommended consolidation ratios.

Strengths and Limitations

- **Best fit:** Bursty notebooks, CI jobs, and trusted development workloads with substantial idle time.
- **Low friction:** It changes advertised resource capacity without requiring application cooperation.
- **Memory:** GKE states that GPU memory limits are not enforced between shared jobs (docs.cloud.google.com).
- **Compute semantics:** Replica count is placement capacity, not a proportional performance entitlement.
- **Telemetry:** DCGM Exporter cannot associate metrics with containers when device-plugin time-slicing is enabled.
- **Operations:** The Operator does not automatically monitor changes to the time-slicing ConfigMap, so configuration rollout needs an explicit reconciliation procedure.
- **Composition:** MIG and time-slicing can be combined, but sharing within a MIG instance retains temporal contention among its replicas.

Full-Device Passthrough

Capabilities

Passthrough assigns an entire PCI device to one virtual machine. The device is not shared with other VMs (docs.openstack.org). This provides a clean ownership model and guest-native driver access, but it consumes a whole GPU and shifts monitoring into the guest. NVIDIA documents that performance monitoring for a passed-through GPU may be available only from within the VM (docs.nvidia.com).

On Linux, Virtual Function I/O (VFIO) exposes direct device access in an Input-Output Memory Management Unit (IOMMU) protected environment (docs.kernel.org). The ownership unit is the IOMMU group, not necessarily one PCI function, because platform topology may prevent device-level granularity (docs.kernel.org).

Adoption and Scheduler Fit

Passthrough belongs primarily to a VM scheduler and hypervisor lifecycle, not a container replica model. OpenStack requires IOMMU on the host operating system (docs.openstack.org). Red Hat similarly notes that a passed-through device becomes unavailable to the host (docs.redhat.com). Hyper-V Discrete Device Assignment requires PCIe Access Control Services in the root complex (learn.microsoft.com).

Strengths and Limitations

- **Best fit:** One VM that requires native access to a whole GPU.
- **Isolation model:** VM and IOMMU boundaries are easier to reason about than cooperative process sharing, subject to platform grouping.
- **Utilization:** Idle capacity cannot be independently scheduled to another VM.
- **Mobility:** Live migration depends on device, platform, and hypervisor support (docs.openstack.org).
- **Memory operations:** Red Hat documents that VFIO assignment pins VM memory and prevents memory ballooning (docs.redhat.com (docs.redhat.com).
- **Mode changes:** Some hypervisors require a host reboot to switch a GPU between passthrough and vGPU modes.

Dedicated GPU: The “None of These” Result

Capabilities

A dedicated GPU is an exclusive physical accelerator assigned to one host workload without a sharing layer. It may run bare metal or be reserved to one container or job. This is the appropriate baseline whenever the cost of interference, operational ambiguity, or recovery coupling exceeds the value of consolidation.

Microsoft’s Windows Server guidance says a workload running directly on the physical host has no need for graphics virtualization (learn.microsoft.com). AWS’s high-performance computing guidance similarly says optimum performance for tightly coupled workloads occurs when compute-node memory is not shared (docs.aws.amazon.com). These statements do not guarantee latency, but they identify when removing a sharing layer is a rational test baseline.

Adoption, Strengths, and Limitations

- **Best fit:** Latency-sensitive serving, large models that consume nearly all memory, tightly coupled training, and workloads requiring whole-device reset control.
- **Operational clarity:** One owner simplifies chargeback, incident scope, and maintenance coordination.
- **Performance baseline:** Dedicated allocation is the control case against which each sharing mode should be compared.
- **Cost:** Idle capacity is stranded unless the scheduler moves the whole device between jobs.
- **Scale:** It may require more accelerators, slots, power, cooling, and network endpoints.
- **Recovery:** A GPU reset requires all applications to release the target device first (docs.nvidia.com), so single ownership reduces the number of affected parties.

Feature Comparison

Table 1 summarizes the resource and isolation semantics. “Bounded” means the mechanism provides a defined hardware partition, not that every workload will meet a latency objective.

Mechanism	Memory isolation	Compute partitioning	Fault boundary	Oversubscription	Process or guest model	Observability and accounting	Likely scheduler fit
MIG	Dedicated memory resources and isolated paths	Spatial profiles with dedicated resources	Hardware-backed between instances	Not inherent, but time-slicing can be layered within instances	Separate containers, processes, or supported virtualized arrangements	MIG-level metrics exist; Slurm gpumem and gpuutil accounting is unavailable	Kubernetes or Slurm, which supports task binding (slurm.schedmd.com)
MPS	Separate client address spaces with limited error containment	Concurrent kernels with configurable usage ceilings, not reserved partitions	Limited error containment	Multiple clients share one GPU	Cooperative, trusted CUDA processes (aws.amazon.com)	Activity can be attributed to the server process	Slurm MPS GRES or application-managed services
Time-slicing	None between replicas (docs.redhat.com)	Temporal interleaving; replica count is not proportional compute	None between replicas	Explicit replica oversubscription	Independent pods or processes that tolerate contention	Container attribution is limited under DCGM Exporter	Kubernetes integer extended resources (kubernetes.io)
Passthrough	Whole device belongs to one VM	Exclusive whole GPU	VM and IOMMU boundary, subject to IOMMU grouping	No	One VM and guest driver stack (docs.openstack.org)	Primarily guest-side monitoring	Hypervisor or VM scheduler

Mechanism	Memory isolation	Compute partitioning	Fault boundary	Oversubscription	Process or guest model	Observability and accounting	Likely scheduler fit
Dedicated GPU	Exclusive device ownership	Exclusive whole GPU	One workload owns the device lifecycle	No	Bare-metal service, container, or batch job	Simplest owner-level telemetry and chargeback	Appropriate baseline for strict latency evaluation (docs.aws.amazon.com)

The table shows why there is no universal ranking. MIG is the only sharing mechanism here whose defining abstraction is a hardware partition. MPS optimizes cooperative concurrency. Time-slicing increases placement density without manufacturing isolation. Passthrough and dedicated allocation sacrifice fractional utilization for exclusive ownership.

Table 2 provides a versioned compatibility evidence table. These entries are verification checkpoints, not a complete support matrix.

Evidence item	Verified as of	Operational implication
Kubernetes GPU scheduling is stable since **v1.26** (kubernetes.io)	September 20, 2026	A vendor plugin still must expose the GPU resource.
Slurm supports MIG devices beginning with **21.08** (slurm.schedmd.com)	September 20, 2026	Partition geometry must be prepared outside Slurm.
MPS supports Linux and QNX hosts (docs.nvidia.com)	September 20, 2026	General-purpose Windows-host MPS designs are excluded.
MPS v3 memory partitioning requires cgroup v2 and CUDA **13.4+** (docs.nvidia.com)	Source dated September 9, 2026	Validate the exact MPS protocol and CUDA branch, not just “MPS enabled.”
GKE MIG support lists GB200, B200, H200, H100, A100, and RTX PRO 6000 (docs.cloud.google.com)	Source dated September 18, 2026	Cloud model coverage does not establish on-premises firmware or driver readiness.
Hyper-V passthrough can accelerate at most **one VM per physical GPU** (learn.microsoft.com)	Microsoft page dated November 1, 2024	DDA is exclusive assignment, not fractional sharing.

The dated rows expose a practical risk: a design can be conceptually correct but unsupported on the selected GPU, driver, CUDA, hypervisor, or orchestrator release. Procurement approval should therefore attach to a tested bill of materials and version set.

Performance and Benchmarks

Public comparisons rarely control for model architecture, request shape, memory pressure, batch size, GPU profile, clocks, driver, and scheduler policy at once. Therefore “MIG is predictable” or “MPS is faster” should not appear as an unconditional procurement claim. The mechanisms control different resources, and observed performance is the result of both those controls and workload behavior.

The most relevant quantitative evidence found for this report is a 2026 study of MPS and MIG tradeoffs. It reports that favorable MPS cases improved performance by up to **30%** and reduced energy by about **20%**, while memory contention worsened performance by around **30%** (arxiv.org) (arxiv.org). The authors attribute MIG's handling of memory contention to hardware isolation (arxiv.org). These figures are study results, not universal multipliers. Buyers should inspect its workloads and hardware before transferring them to another system.

The performance hypotheses to test are:

- **MIG:** tail latency should remain less sensitive to activity in another hardware partition than with whole-GPU time-slicing.
- **MPS:** throughput may rise when several trusted processes individually leave execution capacity idle.
- **Time-slicing:** utilization may rise for bursty tenants, while p95 latency may widen as active process count grows.
- **Passthrough:** guest overhead and IOMMU behavior should be compared with the same workload on bare metal.
- **Dedicated:** this is the interference-free control case, not an automatic guarantee of the application's service objective.

Each hypothesis needs a confidence interval or repeated-run distribution. A single throughput maximum cannot establish isolation, tail behavior, or recoverability.

Data Analysis and Evidence

The documented limits reveal three different notions of "more users." GKE can divide a supported GPU into up to **seven** MIG slices (docs.cloud.google.com). One MPS server supports up to **60** client CUDA contexts per device under the documented default (docs.nvidia.com). GKE allows up to **48** time-sharing containers per physical GPU (docs.cloud.google.com). These figures count hardware partitions, software contexts, and scheduling consumers respectively. They are not interchangeable density metrics.

Kubernetes reinforces the distinction. Extended resources are whole numbers (kubernetes.io), and GKE does not accept fractional GPU requests (docs.cloud.google.com). A device plugin may advertise several integer time-sliced replicas, but that transformation changes schedulable capacity, not the physics of memory or compute. A request for two replicas can still compete with other processes on the same device.

The operational data also show monitoring asymmetry. Slurm can normally track GPU memory and utilization as Trackable Resources (slurm.schedmd.com), yet it documents no gpumem or gpuutil accounting for MIG (slurm.schedmd.com). GKE states that time-sharing and MPS accelerator metrics apply at node level (docs.cloud.google.com). Passthrough moves device monitoring into the guest. Chargeback precision can therefore be a deciding constraint even when compute behavior is acceptable.

No public evidence supports one universal oversubscription ratio. The rational calculation is empirical:

- **Arrival profile:** concurrent requests, burst duration, and idle fraction.
- **Memory headroom:** weights, key-value cache, activations, allocator fragmentation, and framework reserve.

- **Service objective:** p50 and p95 latency, throughput, deadline misses, and queue depth.
- **Failure cost:** number of workloads affected by an out-of-memory event, fatal fault, reset, or node drain.
- **Operational cost:** partition reconciliation, driver maintenance, monitoring gaps, and scheduler configuration.
- **Capital effect:** GPUs avoided only after performance and isolation criteria pass.

This approach prevents a nominal replica count from being converted directly into a purchasing claim.

Validation Checklist and Test Plan

Before testing, record the complete GPU model and form factor, firmware, host operating system, IOMMU grouping, NVIDIA driver branch, CUDA version, GPU Operator or device-plugin version, container runtime, Kubernetes or Slurm version, and desired MIG geometry. For passthrough, also record hypervisor, guest operating system, guest driver, reset support, and migration policy.

Table 3 is a worksheet. It deliberately contains no invented benchmark values.

Test	Method	Record	Pass condition to define before run
Baseline performance	Run one workload on a dedicated GPU with production request shapes	p50 and p95 latency, throughput, GPU and memory utilization, power	Workload-specific service objective
Contention sweep	Increase active co-tenants or MPS clients one step at a time	Same metrics plus queue depth and context count	Maximum acceptable tail-latency and throughput change
Memory pressure	Grow batch, cache, or allocation until one workload reaches OOM	Which workload fails, peer impact, recovery time	Documented failure scope and clean recovery
Cross-workload fault	Trigger an approved non-destructive application failure in one test tenant	Peer errors, device health, process and pod restarts	No impact outside the accepted trust boundary
Accounting	Compare scheduler allocation, DCGM, guest, and application counters	Attribution completeness and reconciliation error	Sufficient evidence for capacity planning or chargeback
Reset and reconfiguration	Drain workloads, change MIG geometry or reset the device, then restore	Disruption window, manual steps, stale resources, rollback time	Repeatable runbook within maintenance objective
Reboot persistence	Reboot a test node and reconcile desired configuration	Time to schedulable state and geometry correctness	Automated return to declared state
Upgrade compatibility	Repeat smoke and contention tests on proposed driver and Operator versions	Functional differences, metric changes, resource names	No unexplained regression

The worksheet makes the decision auditable. For GPU Smith’s adjacent advisory posture, the relevant first-party practice is its published focus on capacity planning, telemetry, failure-mode analysis, and operating procedures (gpusmith.com). The mechanism should be accepted only against the owner’s thresholds, not against a generic utilization slogan.

Implications and Future Directions

The first implication is architectural: **isolation must be purchased and operated explicitly**. A Kubernetes resource object is an accounting unit. MIG is a hardware partition. MPS is cooperative concurrency. Time-slicing is temporal multiplexing. Passthrough is exclusive VM assignment. Treating these as interchangeable “fractions” creates avoidable security and capacity errors.

The second implication is that combinations need their own names and tests. MPS can run over a MIG instance, and time-slicing can expose replicas of MIG resources. In both cases, MIG provides a boundary around the instance while the workloads inside that boundary still share through another mechanism. Operators should diagram both levels and place only mutually compatible trust domains inside the inner sharing layer.

The third implication is operational. Interfaces continue to evolve. Current MPS v3 memory-partitioning requirements and its MIG restriction make protocol version part of the design. Slurm, meanwhile, expects MIG devices to be prepartitioned (slurm.schedmd.com). Platform owners should pin behavior to a dated compatibility record and repeat acceptance tests after driver, CUDA, Operator, runtime, or scheduler changes.

Finally, observability is a design input. If per-container time-slicing metrics or per-MIG Slurm accounting are required for chargeback, a technically functional sharing mode can still be operationally unsuitable. Application-level request, latency, token, and memory telemetry may need to supplement device counters.

Frequently Asked Questions (FAQs)

What is the practical answer to NVIDIA MIG vs MPS?

Choose **MIG** for hardware-partitioned compute and memory between separate workloads on supported GPUs. Choose **MPS** for concurrent, trusted CUDA processes that cooperate and individually underutilize the device. MPS usage ceilings do not reserve dedicated resources, so it should not be used as if it were a MIG profile.

What changes in MIG vs time slicing?

MIG creates spatial hardware partitions with isolated memory paths. Time-slicing interleaves processes on shared hardware and provides no memory or fault isolation between replicas. Time-slicing can produce more scheduler-visible slots, but the replica count is not a guaranteed compute percentage.

How should buyers compare GPU time slicing vs passthrough?

Time-slicing favors density for trusted, bursty container workloads. Passthrough assigns one complete GPU to one VM, provides guest-native control, and prevents another VM from using idle capacity. The decision turns on trust, VM requirements, contention tolerance, monitoring location, and migration constraints.

Is MIG vs GPU passthrough mainly an isolation decision?

It is also a granularity and operating-model decision. MIG divides a supported GPU among several instances for processes or containers. Passthrough gives one VM the full PCI device. Both can provide meaningful boundaries, but only passthrough gives the guest exclusive control of the entire GPU.

What distinguishes MPS vs time slicing GPU sharing?

MPS enables concurrency among cooperating CUDA processes. Time-slicing interleaves GPU processes and does not require them to cooperate. MPS can improve utilization when kernels leave execution capacity idle, while time-slicing is often simpler for unrelated bursty jobs. Neither is a hardware partition.

What is the best GPU virtualization method for AI workloads?

There is no method that is best across workloads. For untrusted tenants, begin with MIG where supported or exclusive devices. For one VM, use passthrough. For one cooperative multi-process application, test MPS. For trusted interactive jobs with idle time, test time-slicing. For strict latency or whole-device control, use a dedicated GPU.

Conclusion

The correct GPU sharing decision follows constraints in a fixed order: trust boundary, memory isolation, fault containment, process or VM model, performance tolerance, observability, and operations. **MIG** is the hardware-partitioned sharing choice. **MPS** is the cooperative-process concurrency choice. **GPU Operator time-slicing** is the oversubscribed temporal-sharing choice. **Passthrough** is exclusive whole-device assignment to a VM. A **dedicated GPU** is the valid result when sharing adds unacceptable interference or complexity.

Platform teams should resist translating scheduler-visible units into performance entitlements. Integer resource requests support placement and quota. They do not create proportional compute, memory limits, or fault boundaries unless the underlying mechanism does so. Likewise, published maxima for partitions, software contexts, and time-sharing consumers describe different layers and cannot be compared as equivalent capacity.

The procurement-safe approach is to shortlist mechanisms with the decision tree, confirm compatibility against dated primary documentation, and run the worksheet on the exact hardware and software stack. Record p50 and p95 latency, throughput, out-of-memory scope, cross-workload fault behavior, utilization, attribution quality, and reset or reconfiguration effects. A sharing design is ready only when it passes those declared acceptance criteria under representative contention.

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/tesla/mig-user-guide/latest/
2. https://docs.redhat.com/en/documentation/openshift_container_platform/4.20/html-single/hardware_accelerators/hardware_accelerators
3. https://docs.nvidia.com/deploy/mps/when-to-use-mps.html
4. https://docs.openstack.org/nova/latest/admin/pci-passthrough
5. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/amazon-ec2-nested-virtualization.html
6. https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/

7. https://docs.cloud.google.com/kubernetes-engine/docs/how-to/timesharing-gpus
8. https://docs.cloud.google.com/kubernetes-engine/docs/concepts/timesharing-gpus
9. https://arxiv.org/abs/2604.22430
10. https://kubernetes.io/docs/tasks/manage-gpus/scheduling-gpus/
11. https://slurm.schedmd.com/gres.html
12. https://gpusmith.com/
13. https://gpusmith.com/articles/en/gpu-cpu-nic-affinity-rank-binding
14. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/introduction.html
15. https://docs.nvidia.com/datacenter/tesla/mig-user-guide/supported-gpus.html
16. https://docs.cloud.google.com/kubernetes-engine/docs/how-to/gpus-multi
17. https://aws.amazon.com/blogs/containers/maximizing-gpu-utilization-with-nvidias-multi-instance-gpu-mig-on-amazon-eks-running-more-pods-per-gpu-for-enhanced-performance/
18. https://aws.amazon.com/blogs/containers/gpu-sharing-on-amazon-eks-with-nvidia-time-slicing-and-accelerated-ec2-instances/
19. https://docs.nvidia.com/deploy/mps/mpsv3-memory-partitioning.html
20. https://docs.nvidia.com/vgpu/19.0/grid-vgpu-user-guide/using-gpu-pass-through.html
21. https://docs.kernel.org/driver-api/vfio.html
22. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html-single/configuring_and_managing_virtualization/index
23. https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/plan/plan-for-deploying-devices-using-discrete-device-assignment
24. https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/plan/plan-for-gpu-acceleration-in-windows-server
25. https://docs.aws.amazon.com/wellarchitected/latest/high-performance-computing-lens/compute-architecture.html
26. https://gpusmith.com/articles/en/llm-inference-hardware-sizing-guide
27. https://docs.nvidia.com/deploy/nvidia-smi/
28. https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/?stream=top
29. https://gpusmith.com/articles/en/nvidia-gpu-form-factors-explained
30. https://gpusmith.com/articles/en/cuda-compatibility-production-upgrade-gate
31. https://gpusmith.com/articles/en/gpu-cluster-acceptance-testing-runbook

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