WOLF Advanced Technology

NVIDIA NVSMI GPU MONITORING & DEBUGGING

– WHITEPAPER

By: Marc Henney
Product Manager, Product Management

WOLF ADVANCED TECHNOLOGY

TABLE OF CONTENTS

EXECUTIVE SUMMARY

NVIDIA's System Management Interface (NVSMI), accessed via the nvidia-smi tool, is a vital utility for monitoring and managing GPU health in real time.

This white paper provides engineers and technical teams with a comprehensive guide to using NVSMI on our WOLF Advanced Technology modules like the WOLF-3476 (XMC-A2000E-VO) used for this example, which features an NVIDIA Ampere RTX A2000 (GA107) GPU.

We cover how to initiate NVSMI, interpret its output, and leverage key commands to observe GPU utilization, temperature, power usage, memory status, and error metrics.

We also demonstrate how these insights can be used to debug performance issues, optimize power/thermal behavior, and ensure reliability in mission-critical aerospace and defense applications.

Real-world use cases from the WOLF-3476 tests illustrate how NVSMI helps connect GPU performance data to practical applications in embedded systems.

By the end of this paper, engineers new to NVSMI will understand how to confidently use this tool to monitor WOLF's GPU modules and will be aware of next steps (like exploring NVIDIA's Management Library APIs) for deeper GPU management capabilities.

INTRODUCTION

The **NVIDIA System Management Interface (NVSMI)** is a command-line tool (invoked by the nvidia-smi command) for querying and controlling NVIDIA GPUs, built on top of the NVIDIA Management Library (NVML).

NVSMI provides a **consolidated view of GPU status** – including metrics like utilization, temperature, clock speeds, power draw, and memory usage – and allows certain management operations with administrator privileges.

NVSMI is supported on both **Linux** and **Windows** systems and is included with the NVIDIA driver installation. This makes it a standard tool for anyone working with NVIDIA GPUs, from data center administrators to embedded system engineers.

WHY NVSMI MATTERS FOR WOLF & OUR CUSTOMERS:

In **mission-critical aerospace and defense applications,** GPUs are often deployed in harsh, SWaP-constrained environments (Size, Weight, and Power constrained) such as ruggedized airborne, land or naval systems.

In these scenarios, **reliability and performance monitoring are paramount.** NVSMI allows engineers to **monitor the GPU's health and activity in real-time,** which is critical for diagnosing issues in the field and ensuring the GPU is operating within safe thermal and power limits.

EXAMPLE;

A WOLF XMC or VPX module with an NVIDIA GPU might be performing **AI inference for ISR (Intelligence, Surveillance, Reconnaissance)** or processing high-speed sensor data. Using NVSMI an engineer can assess if the GPU is being **fully utilized or if bottlenecks exist,** and whether factors like **thermal throttling or power capping** are affecting performance. Quickly catching anomalies, such as an unexpected rise in temperature or ECC memory errors, to **prevent** potential **system failures** and aid in debugging.

Error counts and reliability metrics (like ECC memory errors) is an especially crucial tool for high-reliability applications in aerospace/defense, where radiation or extended operation can cause memory bit flips. **Error monitoring** helps ensure the GPU's outputs can be trusted or alerts engineers when hardware might need attention.

HARDWARE SET-UP

- XMC-A2000E-VO - [WOLF-3476]
- Kontron VX3020 3U VPX SBC (x86)
- 8-Slot Pixus Chassis.
- Monitor
- Keyboard and mouse over USB

CONFIGURATION

The WOLF-3476 module is mounted in the XMC 2.0 slot in a Kontron VX3020 3U VPX SBC (x86) in an 8-Slot Pixus Chassis. Connected to display via internal graphics over DP, connected to keyboard and mouse over USB.

INTRODUCTION & SETUP

One advantage of NVSMI is that it typically **requires no separate installation.** Instead, it comes bundled with the NVIDIA GPU drivers on both Linux and Windows. For the WOLF-3476 module that we used for this example, the appropriate NVIDIA drivers were already installed as part of the software stack.

However, to ensure everything is set up correctly, follow these steps.

- Verify Driver Installation: Ensure that the NVIDIA drivers are installed and loaded.
  - **On Linux:** You can check that the kernel module is loaded (using lsmod | grep nvidia).
  - **On Windows:** Ensure the GPU appears in Device Manager without issues. Usually, if the driver is correctly installed, the nvidia-smi command will be present.

Note: Running nvidia-smi is a good test to confirm the driver is functioning. If you run nvidia-smi and get a valid output (rather than "command not found" or an error), it indicates the driver and NVSMI are set up.

- **Environment Requirements:** NVSMI supports all standard NVIDIA-driver-supported Linux distributions and 64-bit Windows versions.
  - Make sure you're using a supported OS (for example, a recent Linux kernel or Windows 10/11) that's compatible with the WOLF module GPU.
  - Typically, WOLF modules support Linux (e.g., Ubuntu or a Yocto-based distro for embedded) and Windows, and even some RTOS options. For the examples in this paper, we assumed a Linux environment for simplicity.

- **User Privileges:** Basic nvidia-smi queries can be run as a normal user, but **administrative (root) privileges** are required for certain operations (like changing power limits or clock settings).
  - If you plan to modify GPU states (power or clocks), be prepared to use **sudo** on Linux or an Administrator Command Prompt on Windows.

- **Additional NVSMI Install:** Since NVSMI is included with the driver, you do not need to install anything separately. However, if drivers are not installed then:

On Linux: run the following commands

1. Refresh package lists – **sudo apt update**
2. Ask Ubuntu which driver is recommended – **ubuntu-drivers devices**
   This shows for e.g. "nvidia-driver-550 - distro non-free recommended"
3. Install the recommended package (or pick a version) – **sudo ubuntu-drivers autoinstall**
4. Reboot – **sudo reboot**

On Windows:

- Download the latest Game Ready or Studio driver from NVIDIA. (or use the driver package provided by WOLF for the target board if available).
- Install and reboot.
- Open an Administrator PowerShell or CMD and run: nvidia-smi.exe

- **CUDA Toolkit (Optional):** NVSMI is independent of the CUDA toolkit; you do not need CUDA installed to use NVSMI. However, NVSMI will display the CUDA version if one is present. For example, on a system with CUDA, the top of the nvidia-smi output will list both Driver Version and CUDA Version.

INITIATING THE NVSMI CONSOLE

To start using NVSMI on the target system, you need to open a command shell and invoke the nvidia-smi command. Here's how to get started:

- **Open a Terminal or Command Prompt:**
  - On Linux, open a terminal window (for instance, press **Ctrl+Alt+T** on Ubuntu to launch a terminal.
  - On Windows, open the **Command Prompt** or **PowerShell** (preferably with Administrator rights if you plan to change settings).

- **Navigate to the correct Directory (if necessary):** In most cases, nvidia-smi is in your PATH.
  - On Linux, this is typically /usr/bin/nvidia-smi.
  - On Windows, it is usually located in C:\Program Files\NVIDIA Corporation\NVSMI\nvidia-smi.exe.

If for some reason it's not in the PATH on Windows, you may need to navigate to that directory or add it to PATH. On WOLF's Linux builds, the tool should be accessible globally.

- **Run a Test Command:** Simply type **nvidia-smi** and press **Enter.**

This will query the GPU status and print a summary to the console. If the system has multiple NVIDIA GPUs, nvidia-smi will show all of them by default. You should see a table output with the GPU name, utilization, memory usage, temperature, etc. If instead you see an error, it may indicate an issue (for example, no NVIDIA driver running or the GPU not being detected). Ensure the WOLF board is powered, and the GPU drivers are loaded.

- **Persistence Mode (for Embedded Systems):** If you are in an embedded Linux environment, you might want to enable persistence mode **(nvidia-smi -pm 1)** to keep the GPU driver loaded and responsive between uses.

This is not strictly necessary for using NVSMI, but it can avoid delays in re-initializing the GPU for each command on headless systems. On a development workstation, this is usually not an issue, but on some headless deployments it can be useful.

By following these steps, you have essentially "booted up" the NVSMI interface. All subsequent commands will likewise run in the terminal. NVSMI outputs can be refreshed periodically by re-running commands or using built-in looping options (e.g., the --loop flag or using Linux's watch command) for continuous monitoring.

COMMAND-BY-COMMAND GUIDE

NVSMI (nvidia-smi) offers a variety of commands and options to retrieve different aspects of GPU information. In this section, we provide a command-by-command guide to the most relevant NVSMI commands for GPU monitoring and debugging, especially as they apply to the WOLF-3476's NVIDIA GPU. For each command, we explain what it does, how to use it, and why it's useful in an embedded aerospace/defense context.

nvidia-smi: Basic GPU Status (Utilization, Temperature, Memory, Drivers)

The base command **nvidia-smi** (with no arguments) is the first tool to reach for to get an **overview of the GPU's status.** Running nvidia-smi prints a snapshot summary of each GPU in the system, including:

- **Driver and CUDA Version:** At the top, it displays the NVIDIA driver version and the CUDA version (if applicable). This helps verify that the correct driver is installed and can be important when matching software compatibility.

- **GPU Name and Index:** It will list each GPU by index (GPU 0, GPU 1, etc.) and the product name (e.g., "NVIDIA RTX A2000"). In our case with WOLF-3476, you'll see the GPU corresponding to the Ampere A2000E module.

- **Temperature and Fan:** The current core temperature of the GPU (in Celsius) is shown, and fan speed if available. In our example, you can see the **Temp: 37°C** indicating the GPU is at 37°C. (Note: Some embedded modules may show "N/A" for fan if there's no directly controllable fan on the GPU).

- **Performance State (P-State):** A readout labeled **Perf** (performance state) indicates the current **P-state** (P0 being max performance, P8/P12 etc. being lower power states). At idle, the GPU will usually be in a high-numbered P-state (lower performance) to save power. In this example Perf is P8.

- **Power Usage/Capacity:** Shown as **Pwr: Usage/Cap,** this field displays the real-time power draw of the GPU and the current power cap. In our example, at idle, you can see a low usage of 3 W / 26 W (meaning the GPU is drawing 3 watts out of a 26 W power limit). Under load, the "Usage" will increase and could approach the cap. This is crucial for monitoring whether the GPU is within power budgets.

- **Memory Usage:** Typically labeled as **Memory-Usage**, it shows the **GPU memory consumption** as Used/Total. In this instance Mem-usage is 8 MiB / 8192 MiB would meaning 8 MB of VRAM is in use out of an 8 GB total.

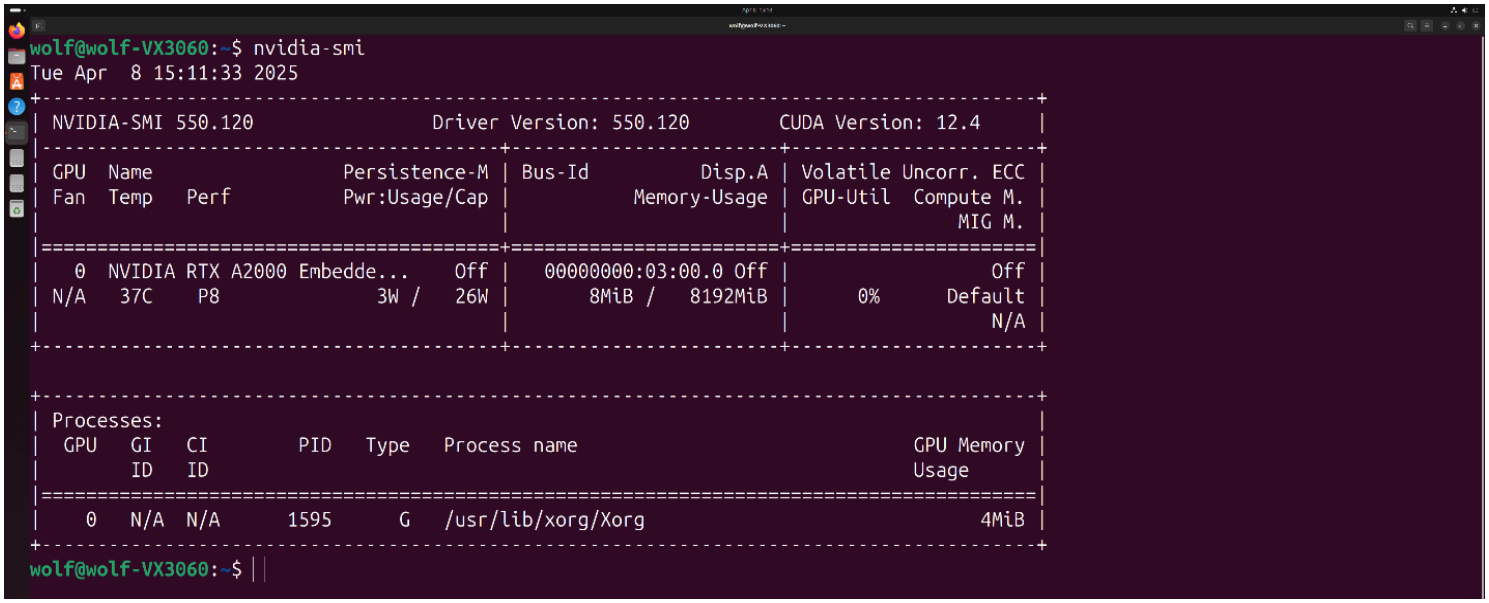
- **GPU Utilization:** Often labeled **Gpu-Util,** it shows the current percentage of the GPU's compute engines in use. An idle GPU will show 0% utilization, while a fully loaded one should show near 100%. This helps identify if the GPU is busy or waiting.

- **Other Utilization Metrics:** Depending on the GPU, other usage like Encoder (Enc) or Decoder (Dec) utilization may be shown (especially if video encoding/decoding is happening). Those remain at 0% if not in use.

- **ECC Status:** If the GPU supports ECC (Error-Correcting Code memory), the summary might indicate if ECC is enabled or disabled. (On some GPUs ECC can be toggled, but it often requires a reboot to enable/disable.)

- **Compute Mode:** This indicates if the GPU is in default mode or a restricted compute mode (used in multi-user environments, usually "Default" for our purposes).

- **Active Processes:** At the bottom of the output, nvidia-smi lists any **processes using the GPU** along with their PID, process name, and memory usage. For example, on a system with a desktop environment, you might see the Xorg server or GNOME Shell using some GPU memory. In our WOLF-3476 test, we saw the Xorg process using ~4MiB, and no heavy compute processes, indicating the GPU was largely idle.



USE CASE: The default nvidia-smi output is a quick health check. For example, comparing an **idle vs. loaded state** is instructive. At idle, you might see low utilization (0–1%), low power draw, and clocks at base levels (P8/P12). When you start a workload, say a neural network inference, you'd observe GPU Util% rising towards 90-100%, memory usage increasing (if the model is loaded into VRAM), temperature climbing steadily, and power usage rising towards the cap. The **difference between idle and load** tells you how the GPU behaves.

In our example, we ran a sample workload on the WOLF-3476 and observed power consumption jumping from ~3 W to near 20 W and GPU utilization going from 0% to ~98%, illustrating the GPU handling the workload. This data helps us understand **when the GPU is the bottleneck.**

BENEFIT: This makes it easy for engineers to quickly assess whether an application is efficiently utilizing the GPU, or if there's available headroom to run additional tasks. It also provides at-a-glance insights into environmental conditions such as temperature and power, helping ensure the module stays within safe operating limits.

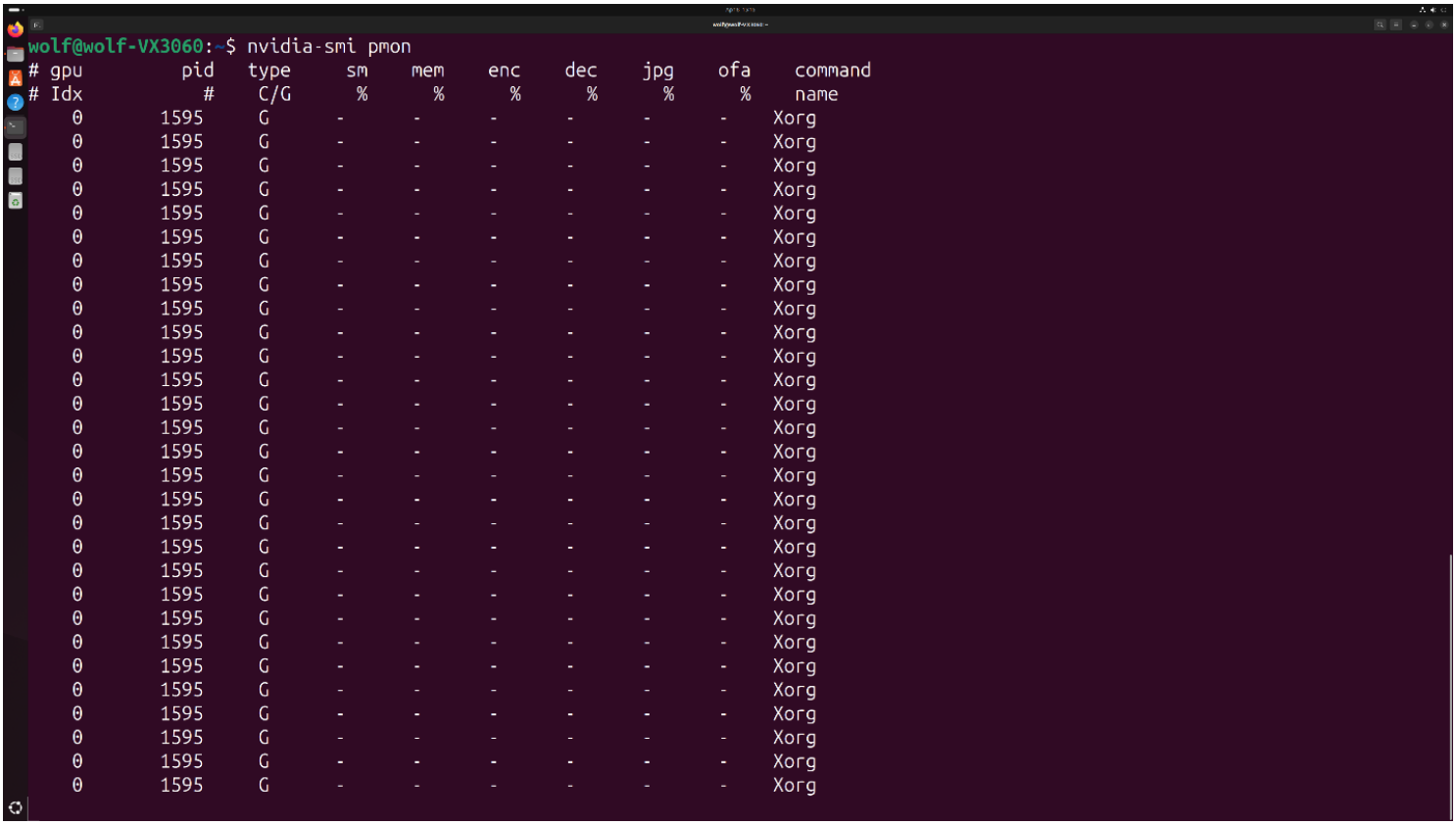
nvidia-smi pmon: Monitoring Active GPU Processes

When you need continuous monitoring of GPU usage per process, nvidia-smi pmon (process monitor) is the go-to command. This tool provides a live, scrolling view of all processes utilizing the GPU, updating every second by default. Each line of output corresponds to a process on a GPU and includes fields such as:

- **GPU index:** Which GPU the process is running on (useful if multiple GPUs).
- **PID:** The process ID of the application using the GPU.
- **Type:** The context type, typically "C" for compute (CUDA) processes or "G" for graphics (like display servers). In our test, we saw the Xorg process with a type "G" on GPU 0.
- **SM Utilization (%):** The streaming multiprocessor usage of that process, as a percentage (this is the compute engine utilization for that process since last sample).
- **Memory Util (%):** The percentage of GPU memory bandwidth the process is using.
- **Encoder/Decoder Util (%):** If the process is using NVENC or NVDEC (video encoder/decoder), those utilizations are shown.
- **Process Name (Command):** In some modes, the command name is shown so you know which application it is (in default pmon, you see the PID and can cross-reference it, but there are options to show names).

By default, nvidia-smi pmon runs in an **interactive loop** (until you press Ctrl+C) and prints the above info continuously. If no process is using the GPU at the moment, it will either show blank or "–" for metrics until something appears.

HOW TO USE IT:



Simply run nvidia-smi pmon with no arguments to monitor all GPUs. You can also target specific GPUs (e.g., nvidia-smi pmon -i 0,1 to monitor GPU0 and GPU1 only). The output refreshes each second. This is useful to leave running in a window while you start and stop GPU tasks to see them come and go in real time.

USE CASE: In an **embedded system scenario**, you can use nvidia-smi pmon to verify that multiple applications (or containers) are sharing the GPU properly. For instance, if you have a graphics application (like a map rendering) and an AI inference service running together on the same GPU (common in some mission systems), pmon will show both processes and how they divide the SM (compute) and memory resources.

BENEFIT: **Process monitoring** helps engineers understand how **GPU resources are distributed across applications,** ensuring that critical workloads receive the compute time they need. The nvidia-smi pmon command provides real-time visibility into GPU utilization, making it easy to spot issues like runaway processes consuming more resources than expected or an important task not using the GPU as intended. This level of **insight** is especially valuable when **running concurrent workloads** on a single GPU

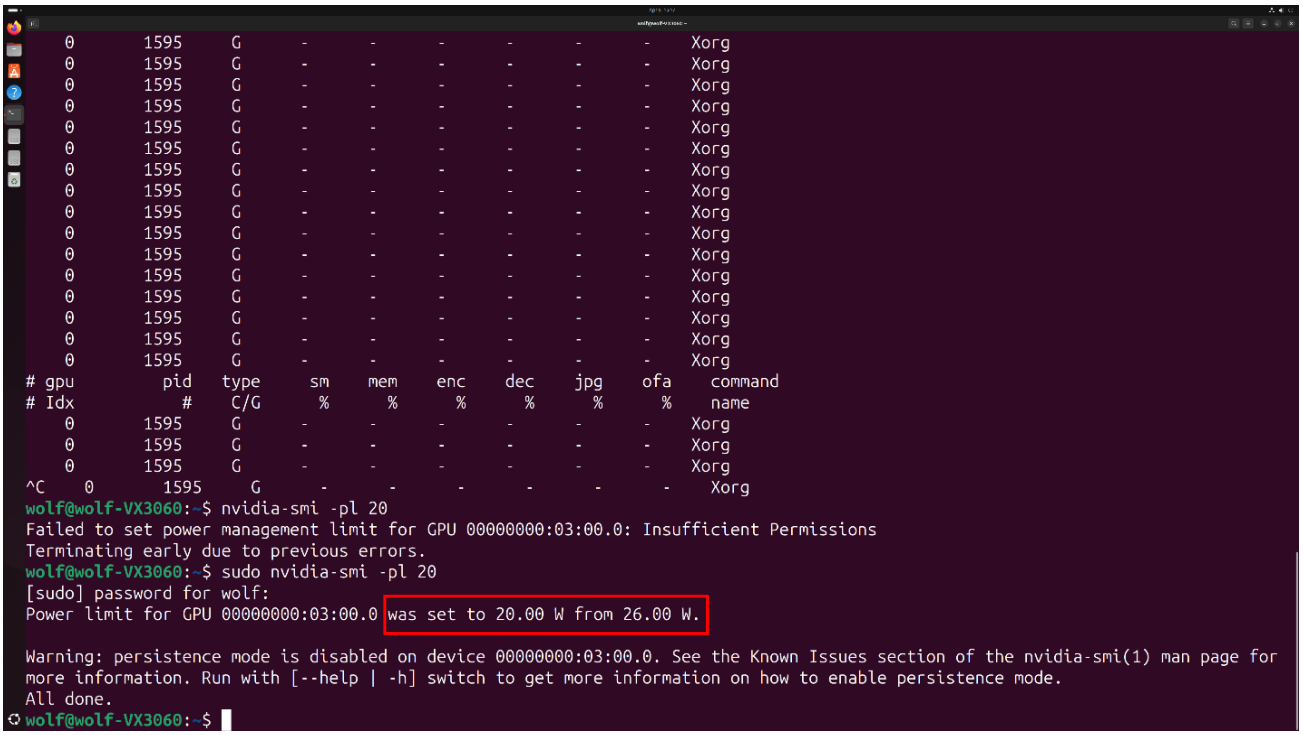
nvidia-smi -pl <WATTS>: Power Limit Control and P-State Management

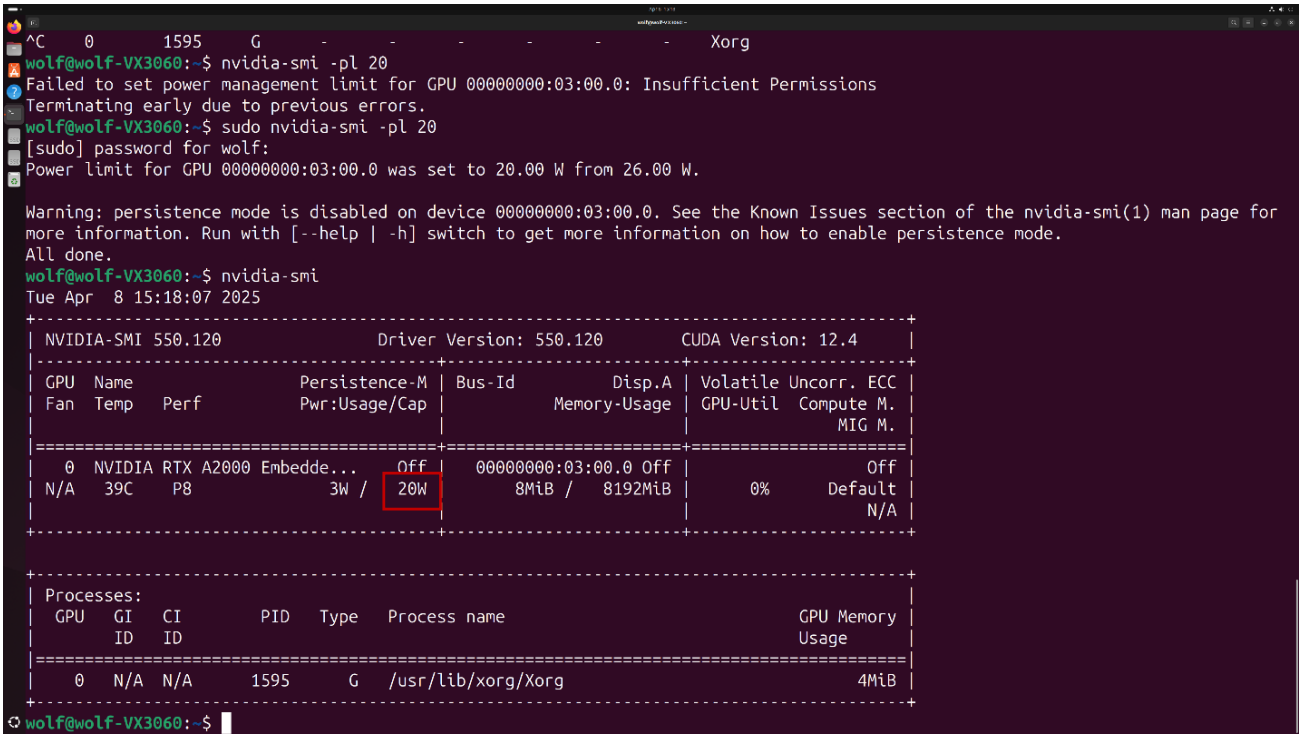
The **power limit** command, nvidia-smi -pl <value>, allows administrators to **set a maximum power draw (in Watts)** for the GPU.

This is a powerful feature for controlling the GPU's thermal and power footprint, especially important in SWaP-constrained systems. **Changing the power limit requires root/administrator privileges** because it directly affects hardware behavior.

HOW TO USE IT: In our example, using **sudo nvidia-smi** -pl 20 on our WOLF-3476's GPU (Ampere A2000E), changed power limit from the default 26 W to 20 W. After setting a new power limit, the change takes effect immediately, throttling the GPU's power usage to not exceed the new cap.

You can find the minimum and maximum allowed power limit by running **nvidia-smi -q -d POWER** (This will show "Min Power Limit" and "Max Power Limit").





Effect on Performance (P-States): The GPU's **P-state (performance state) and clock speeds** will adjust in response to a power cap. If you set a very low power limit, the GPU will likely drop to a lower P-state when under load, capping its clocks to stay within the wattage. In our example when we reduced the cap from 26 W to 20 W, the GPU under load could no longer reach the highest performance level; its clock speeds were reduced, and it stayed in a lower performance state while drawing at most 20 W.

This can **reduce performance** (throughput or frame rates), **but it also reduces heat output** and power consumption. Conversely, raising the power limit (up to the max allowed) ensures the GPU can use its full performance headroom (important if the environment can cool it sufficiently).

USE CASE: In aerospace and defense systems, we often have strict power budgets. A VPX chassis might allocate only so much power per slot. If our WOLF GPU card can technically draw 80 W at full tilt but the system can only provide, say, 60 W safely, we would set the power limit to 60 W to prevent overload. Another use-case is thermal management: in a hot environment where the cooling is limited, capping power can keep the GPU from generating more heat than the system can dissipate, thus avoiding thermal shutdown or throttling. It's also a way to explore **power/efficiency trade-offs** – e.g., find how performance is affected if we limit a GPU to 75% of its max power.

BENEFIT: By learning to use power limits, engineers can apply **power efficiency strategies** that are essential for SWaP-optimized systems. Intentionally capping power allows teams to evaluate how much performance is traded for power savings—critical insight when designing for platforms like UAVs or airborne systems, where every watt counts. Combining this with an understanding of P-states also helps explain scenarios where a GPU doesn't reach full clock speeds (e.g., "It's in P2 state due to the 60 W power cap set for this application").

Overall, -pl is a tool for balancing **performance vs. power** to meet the requirements of rugged deployments.

nvidia-smi dmon -s: Real-Time Monitoring of PCIe Throughput, Power, and Thermals

1.1. nvidia-smi dmon –s p

NVSMI includes a device monitoring mode invoked by nvidia-smi dmon, which provides a real-time, line-by-line update of various GPU metrics. The -s option allows you to select specific metrics to monitor. In our use, we focus on two kinds of metrics: power/thermal and PCIe bus utilization.
The command nvidia-smi dmon -s p will display a live table of power and temperature readings for each GPU, updating every second. (Here p stands for the "Power" metric group, which includes power usage and temperature). The output shows, per sample, columns such as GPU index, power draw (Watts), and GPU temperature (°C). Running dmon -s p on the WOLF-3476 while starting a GPU-intensive task shows the power usage climbing and the temperature rising slowly until the cooling stabilizes it, all in real-time text updates. In our idle test, dmon -s p showed the GPU drawing ~3 W at 40°C steadily. If we were to apply a load, you'd see those numbers update to higher values live.

1.2. nvidia-smi dmon –s t

You can use the command nvidia-smi dmon -s t (for "throughput") which shows PCIe Rx/Tx utilization in MB/s). to monitor PCIe bandwidth usage – i.e., how much data is flowing to/from the GPU over the PCI Express bus – because in high-speed embedded systems, the PCIe interface can be a bottleneck if saturated.

```
wolf@wolf-VX3060:~$ sudo nvidia-smi -pl 26
Power limit for GPU 00000000:03:00.0 was set to 26.00 W from 20.00 W.

Warning: persistence mode is disabled on device 00000000:03:00.0. See the Known Issues section of the nvidia-smi(1) man page for
more information. Run with [--help | -h] switch to get more information on how to enable persistence mode.
All done.
wolf@wolf-VX3060:~$ nvidia-smi dmon -s p
# gpu    pwr  gtemp  mtemp
# Idx      W      C      C
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
    0      3     40      -
```

By running **nvidia-smi dmon -s t,** you can watch how much data each GPU is sending/receiving over PCIe each second (supported on newer GPUs). For clarity, one could combine or run separate instances to see both power and PCIe (though NVSMI itself might not combine them in one command, you'd run two separate monitors or specify multiple letters like -s pt to show both groups if supported).

USE CASE: Many of our systems involve streaming large volumes of data (e.g., sensor feeds or video frames) to the GPU for processing (sometimes via GPUDirect RDMA from an FPGA or NIC). If the GPU isn't getting data fast enough, the PCIe bus could be a limiting factor. By observing PCIe throughput, we can determine if we are close to the theoretical limit of PCIe Gen3 or Gen4. For instance, a Gen3 ×8 link has ~7.88 GB/s each direction; if our application is consistently hitting, say, 7000 MB/s (7 GB/s), we know we are saturating near the bus capacity.

Upgrading to Gen4 (which doubles bandwidth) or optimizing data transfers might then be considered. In multi-GPU setups or GPU-to-GPU (NVLink) scenarios, similar throughput monitoring is available (NVLink usage can also be shown in some NVSMI queries, though not on the A2000 since it doesn't have NVLink). In summary, with appropriate flags, nvidia-smi dmon gives a continuous view of what's happening with power, thermals, and data transfer rates. It runs until stopped, allowing engineers to observe trends over time, not just snapshots.

BENEFIT: Understanding **PCIe bandwidth utilization** is essential for AI inference and ISR workloads, as it helps **identify performance bottlenecks caused by data movement rather than GPU compute limitations.** The dmon command allows engineers to verify that the PCIe interface is operating as expected.
For example, revealing reduced throughput after moving a card from a Gen4 to a Gen3 slot. Monitoring bandwidth usage also supports design decisions around interconnects; if an application clearly benefits from higher throughput, that insight directly informs hardware selection and optimization. On the power and thermal side, continuous monitoring enables more effective cooling validation by tracking how quickly temperatures stabilize or whether power throttling occurs over time.

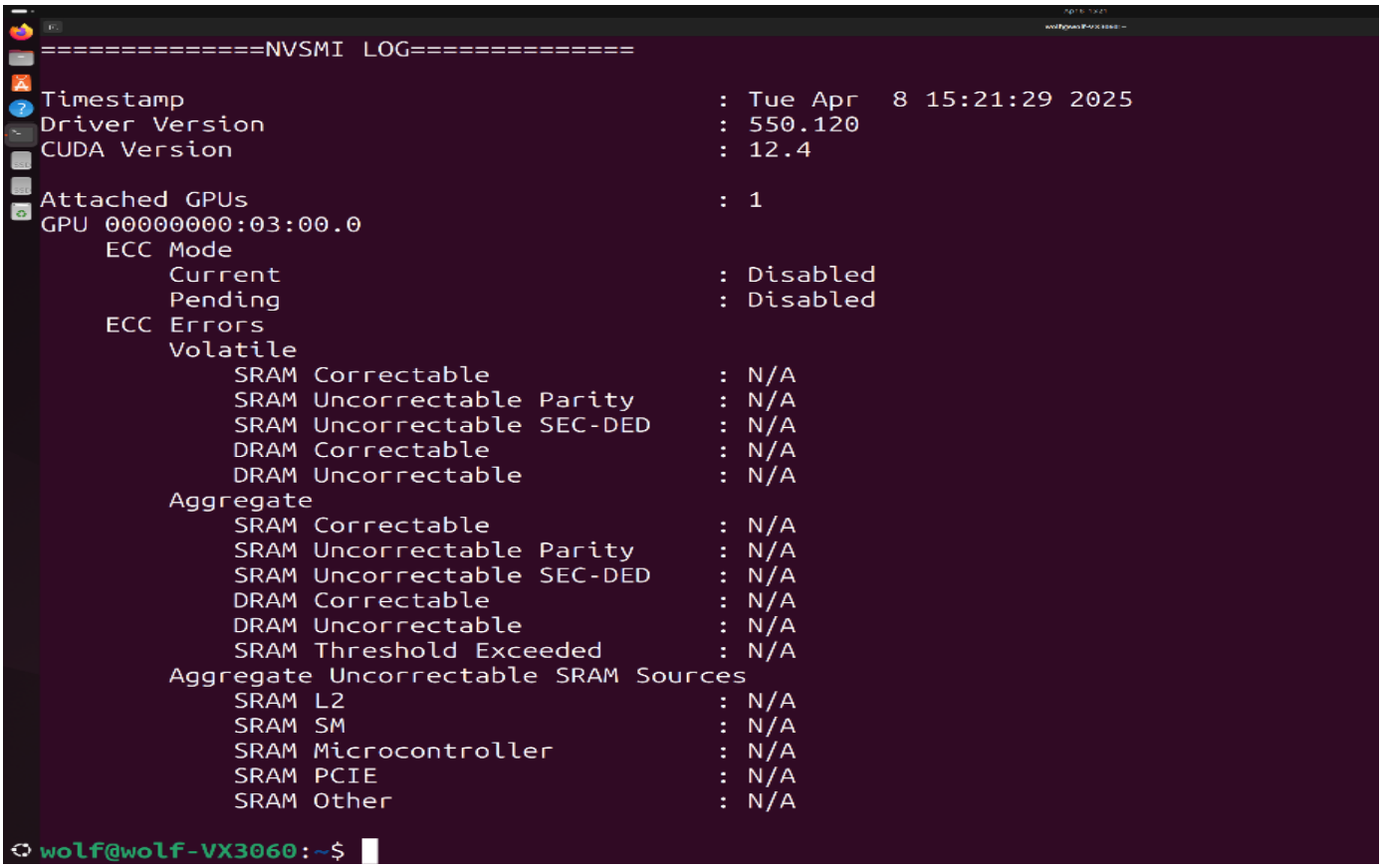
nvidia-smi -q -d ECC: Checking GPU Reliability Metrics (ECC Errors)

ECC (Error-Correcting Code) memory is a feature that detects and sometimes corrects memory errors on the fly. NVIDIA's professional and data center GPUs often support ECC on their VRAM. The command **nvidia-smi -q -d ECC** queries the GPU for detailed ECC status and error counts. This includes the number of single-bit and double-bit memory errors that have occurred, both volatile (since last driver load) and aggregate (lifetime). It also indicates whether ECC is currently enabled on the GPU.

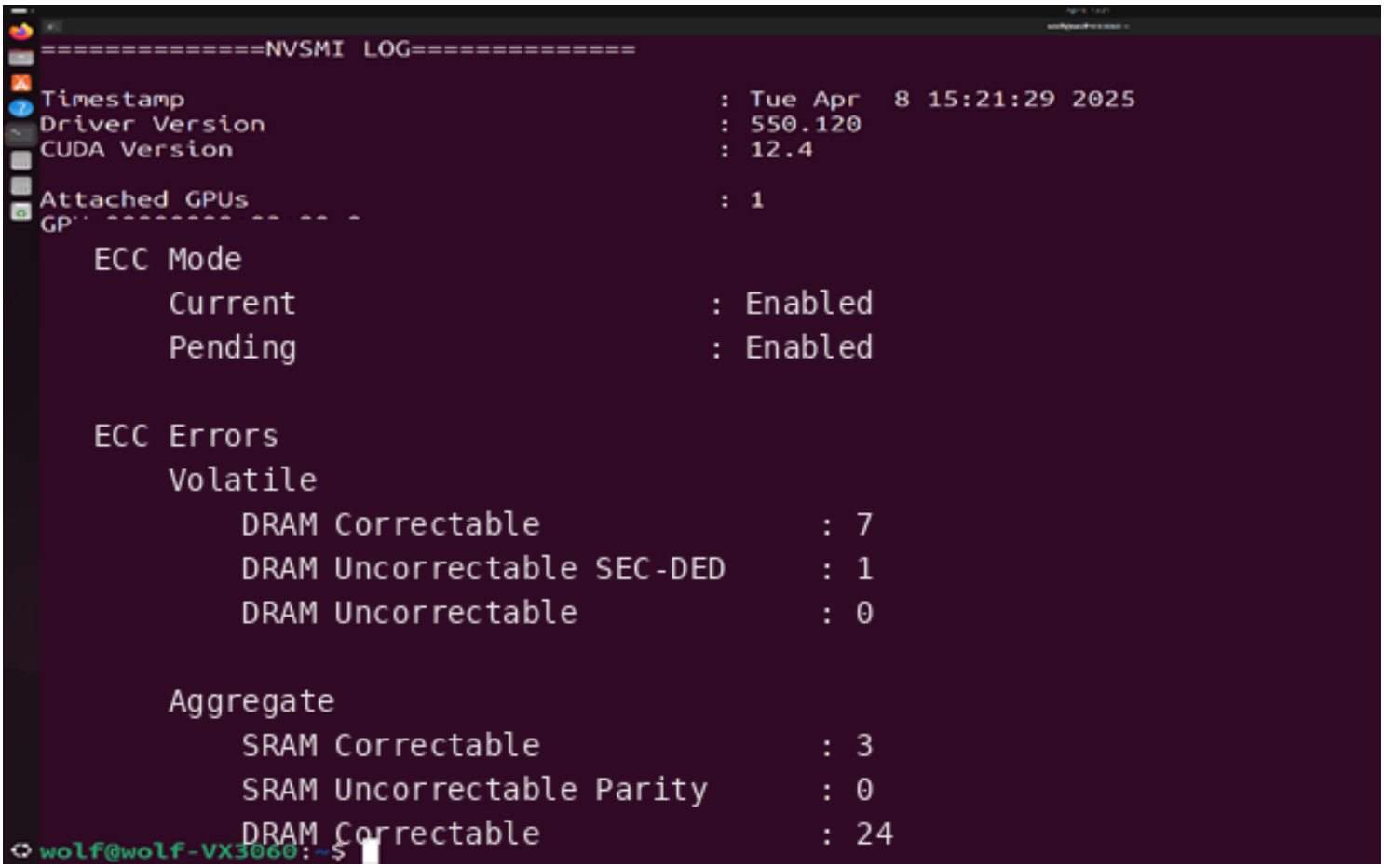
Running nvidia-smi -q -d ECC will produce a detailed readout. Below are the key information provided:

- **ECC Mode:** Whether ECC is On or Off. Many GPUs default to ECC off (for example, GeForce and some embedded versions might not support ECC or have it disabled for performance reasons). Quadro and Tesla GPUs often allow enabling ECC. On our WOLF-3476's RTX A2000E, we found that ECC mode was **disabled** by default.
- **Single-Bit Errors (Correctable):** Count of corrected errors, often labeled as "Volatile ECC Single-Bit Errors" and "Aggregate ECC Single-Bit Errors". A non-zero count here means ECC caught memory bit flips (which were corrected, preventing bad data)
- **Double-Bit Errors (Uncorrectable):** These are more severe because ECC can detect but not correct 2-bit errors in the same word. **Any non-zero here is a red flag,** as those errors mean data was corrupted.
- **VRAM scrub status:** Some GPUs periodically scrub memory for errors; NVSMI may show if that's happening.
- **Row Remapping events:** In some cases (on data center GPUs), if a memory cell is bad, the GPU can remap it; NVSMI would note if such events occurred.

If ECC is disabled or not supported, many of these fields will show **N/A or 0.** In our example, since ECC was off, the output essentially indicated no data (all N/A), which is expected.



For example: This is what it would look like if ECC was enabled and had captured some Errors



USE CASE: For **mission-critical use cases,** monitoring ECC is important because it gives insight into **hardware reliability** over time. If a particular board starts logging ECC errors, it could indicate deteriorating memory modules or a need for additional radiation shielding if used at high altitude. It can also be used as a debugging tool if an application is crashing or producing inconsistent results – one possible culprit could be uncorrected memory errors. While this is rare, having ECC gives an extra layer of trust. WOLF's modules being deployed in defense systems means we care about every bit of data integrity.

Using this command in practice might be something done as part of maintenance or testing: e.g., after running a long mission or a stress test, run nvidia-smi -q -d ECC to ensure there were zero ECC errors (or that only

BENEFIT: Understanding how to query ECC status helps engineers **evaluate GPU reliability** in mission-critical systems. In environments like aerospace or defense, being able to demonstrate that a **GPU hasn't experienced memory** errors over the course of a mission can be essential. ECC reporting tools allow teams to provide clear evidence of system health—or identify accumulating errors early, enabling proactive hardware replacement or system adjustments. While enabling ECC may slightly reduce performance and increase memory usage, the added fault tolerance is often a worthwhile trade-off for high-reliability applications.

ADDITIONAL TOOLS: GPU CLOCK CAPPING & FREQUENCY CONTROLS

Beyond the primary commands above, NVSMI provides additional controls that can be useful for testing and tuning. One such capability is **GPU clock frequency capping.** This refers to limiting the GPU's core (and memory) frequencies to a certain level, independent of (or in addition to) the power limits. For instance, with modern GPUs, you can use nvidia-smi to **lock the GPU clock frequencies** at a desired level.

The command **nvidia-smi -lgc <minFreq>,<maxFreq>** allows you to set a range for the graphics clock. If you specify the same value for min and max, you effectively lock the clock to that frequency.

For example, **sudo nvidia-smi -lgc 1000,1000** would try to lock the GPU core clock at **~1000 MHz.** Similarly, -lmc can lock memory clocks. These operations require admin privileges and are often only supported in persistence mode or certain driver settings. There's also an older method on some GPUs using "application clocks" (nvidia-smi -ac <memClock>,<graphicsClock>) which defines max clocks for applications.

USE CASE: When would you use clock capping? One scenario is thermal testing: by locking the GPU at a high frequency and idle workload, you can increase power draw somewhat steadily to test cooling. Another scenario is to simulate a lower-grade GPU or to create a deterministic performance level.

For example, you might lock the GPU at a lower clock to see if an application still meets real-time requirements – if it does, you've found a way to reduce power usage with minimal performance impact. Conversely, locking at max clock can sometimes eliminate variability when benchmarking. By capping the clock and then running a workload, one can show the difference in performance or power. After removing the cap (nvidia-smi -rgc resets the GPU clocks to default), the GPU can return to boosting to higher frequencies if thermal/power headroom allows.

It's important to note that **GPU Boost (dynamic clock scaling)** will automatically raise or lower clocks based on load, power, and temperature. So manual clock locking is usually only needed for specific debugging or testing purposes. In normal operation, we rely on the GPU's internal algorithms to manage frequency for optimal performance.

However, NVSMI gives us the levers to override those if needed, which can be invaluable when trying to diagnose an issue. For example, if we suspect that high clock fluctuations are causing an intermittent issue, we could lock the clock to see if the issue goes away.

Summary of additional controls [2]:

- **nvidia-smi -pm {0|1}** – Disable/enable persistence mode (for keeping context).
- **nvidia-smi -lgc <min,max>** – Lock GPU core clocks to a range
- **nvidia-smi -lmc <min,max>** – Lock memory clocks
- **nvidia-smi -ac <memClock,coreClock>** – Set application clocks (max clocks) on supported GPUs.
- **nvidia-smi -q -d CLOCK** – To query supported clocks and current clocks.
- **nvidia-smi --reset-gpu-clocks /** --reset-applications-clocks – To unlock clocks back to default

For engineers, knowing these capabilities means having a full toolbox for GPU performance shaping. While you may not use clock capping in everyday deployment, it can help in a controlled lab setting to mimic different scenarios or to comply with a special requirement (for instance, if a customer needs the GPU not to exceed a certain frequency for validation reasons). It's also a teaching tool – by capping clocks or power, one can demonstrate to a new engineer how performance scales down, reinforcing understanding of concepts like P-states and GPU Boost.

REAL-WORLD APPLICATIONS OF NVSMI ON WOLF GPU MODULES

The commands and data provided by NVSMI become truly meaningful when applied to real-world use cases. In WOLF's typical aerospace and defense deployments, here's how NVSMI assists various stakeholders:

1. Performance Debugging (Frame-rate or Throughput Shortfall)
Problem: An application on a WOLF-3476 with an NVIDIA A2000E is not reaching the expected frame rate or data-processing throughput.

NVSMI-driven solution

1. Run **nvidia-smi** (one-shot) or watch **-n1** nvidia-smi for a live view.
2. Check:
    a. **GPU-Util** – Is it near 100 %?
    b. **Perf** state – Is it **P0** (full clocks) or a lower power state?
    c. **Pwr:Usage/Cap** – Is power pegged at the cap?
3. Interpret:
    a. **Low GPU-Util (≈ 50 %) + High CPU load** → CPU/I-O bottleneck; refactor code to feed the GPU faster.
    b. **GPU-Util ≈ 100 % + Pwr at cap + Perf ≠ P0** → power-throttled; optimize kernels or raise the board's power limit (**nvidia-smi -pl <watts>**).
4. Confirm fixes by re-running the workload and verifying higher GPU-Util with clocks staying at rated max.

In our real case during internal tests, we observed that enabling a certain GPU feature increased power draw to the cap and the GPU clock dropped (power constrained), which we caught by noticing the **Perf state was not P0** and **power was exactly at the limit** – confirming a power throttle situation.

2. Thermal-Throttling Investigation
Problem: In a rugged enclosure the GPU temperature creeps toward its thermal limit (≈ 85 °C) and performance drifts downward.

NVSMI-driven solution

- Command: **nvidia-smi -q** (detailed), optionally watch –n1 nvidia-smi for a live view
- Look for:
    - **Temperature** near limit.
    - **Throttle Reason = Thermal** or clocks markedly below clocks.max.
- Mitigation: improve chassis airflow, adjust fan curves, or reduce load; verify by observing clocks return to full rate and "Throttle Reason" clearing.

By monitoring temperature and clocks, engineers can determine if the GPU is throttling due to heat. For instance, you might see GPU util at 70% but clocks stuck at a lower frequency and a note "Throttle Reason: Thermal" in the full query.

3. Multi-GPU Resource Allocation
Problem: With multiple GPUs (e.g., dual-GPU 6U VPX), workloads are accidentally landing on the wrong device—one GPU overloaded, the other idle.

NVSMI-driven solution

- Command: **nvidia-smi pmon -c 3 or** plain nvidia-smi to list active processes.
- Confirm each PID's **GPU-id** matches the intended mapping (e.g., GPU0: AI inference, GPU1: video encode).
- If mis-routed, update CUDA_VISIBLE_DEVICES, MIG partitions, or container runtime settings; re-check pmon to confirm balanced utilization.

Using nvidia-smi pmon or the default output showing processes, a system integrator can verify that each GPU is assigned the correct processes. If GPU1 is supposed to handle video encoding and GPU0 is running AI inference, they can confirm via NVSMI that the processes are indeed on the intended GPUs and each GPU's utilization is as expected. This prevents scenarios where one GPU is overloaded, and the other is idle due to a misconfiguration.

4. Power-Budget Validation during System Integration
Problem: Integrating a WOLF module into a mission computer requires guaranteeing the GPU never exceeds a 60 W envelope.

NVSMI-driven solution

- Stress test workload while running watch -n1 nvidia-smi --query-gpu=power.draw --format=csv, noheader,nounits.
- If Power Draw > target, cap it with nvidia-smi -pl 60.
- Re-run test and show stakeholders the capped draw staying ≤ 60 W; document the setting in integration notes.

NVSMI allows measurement of **actual power draw under various workloads.** For example, the team can run a worst-case algorithm on the GPU and observe that it draws, say, 55 W peak. If the system budget was 60 W for the GPU, we know we're within limits. If it was drawing 70 W, that's a problem – but we can then use nvidia-smi -pl 60 to cap it. This information can be fed back to the hardware design team or the customer to adjust expectations or cooling.

5. PCIe Throughput Analysis for High-Rate Data Streams
Problem: ISR application sees lower-than-expected end-to-end latency; Potential problem may be with PCIe saturation between FPGA and GPU.

NVSMI-driven solution

- Command: **nvidia-smi dmon -s t** (Tx/Rx MB/s).

Compare measured values with lane limit (e.g., Gen3 ×8 ≈ 7 850 MB/s).

- If at limit → relocate board to Gen4 slot, widen lane count, or compress/partition data.
- If well below limit → investigate algorithmic or a different bus bottleneck.

In this case if performance is lower than expected, engineers might suspect the PCIe link is saturated. Using nvidia-smi **dmon -s t,** engineers can monitor the PCIe Tx/Rx and see if it's hitting the maximum (for Gen3 ×8, around 7850 MB/s). If yes, that explains the bottleneck – the GPU is starved/waiting due to bus limits. The solution might be to move the GPU to a Gen4 slot or compress data more. If not saturated, the bottleneck lies elsewhere.

6. Reliability Monitoring & Predictive Maintenance
Problem: Long-life deployments (satellites, HALE UAVs, 24 × 7 ground stations) must detect early signs of memory degradation.

NVSMI-driven solution

- Scheduled job: **nvidia-smi -q -d ECC** → log **Correctable/Uncorrectable ECC error counters.**
- Trend analysis: rising correctable ECC on a particular board, flag it for proactive replacement during the next maintenance window before an uncorrectable error causes downtime.

Monitoring trends in correctable ECC errors increasing, might indicate a degrading memory module or simply the cumulative radiation exposure. Maintenance schedules can include NVSMI checks to decide if a board should be swapped out. While this might be more relevant to data center, it does apply to defense: e.g., a ground station that's always on can be monitored remotely using NVSMI outputs for signs of hardware issues.

7. Explaining GPU Behavior to Non-GPU Experts
Problem: Field engineer needs to illustrate why performance dropped on a customer system.

NVSMI-driven solution

In this example; the field support engineer would run nvidia-smi on the customer's system and include the output in a support ticket to illustrate that "GPU was overheating (90°C) and downclocking, which is why performance dropped." The straightforward text output can be easier to share than complex performance counters. It provides an **auditable record** of the system state at a given time.

- Capture point-in-time report: nvidia-smi -q > gpu_state.txt.
- Attach file to support ticket or presentation; highlight key lines (e.g., "Temperature = 90 °C, Throttle = Thermal").
- Textual output is self-contained, easily emailed, and reproducible—building customer confidence.

8. Automated Embedded Debug Scripts
Problem: Remote systems need a one-command snapshot when an anomaly is detected.
NVSMI-driven solution

- Internal script (example):

bash
Copy

```
#!/bin/bash
ts=$(date +%Y%m%d_%H%M%S)
nvidia-smi > /tmp/gpu_$ts.txt
nvidia-smi -q >> /tmp/gpu_$ts.txt
nvidia-smi pmon -c 3 >> /tmp/gpu_$ts.txt
scp /tmp/gpu_$ts.txt support@gse.wolf.com:/logs/$HOSTNAME/
```

- Run via SSH or triggered by a monitoring agent; engineers examine the consolidated log to acceler ate root-cause analysis.

This script that captures: nvidia-smi, nvidia-smi -q, nvidia-smi pmon -c 3 (3 samples of process info), etc., could be run when a problem is detected to gather a snapshot for analysis. This could even be run remotely via SSH on a deployed system to assist in remote debugging.

In all these cases, NVSMI acts as the eyes and ears on the GPU. Without it, we'd be "flying blind" regarding the GPU's internal state. With it, we can **correlate system-level behavior with GPU metrics** – perhaps the most important aspect for performance tuning and debugging.

SUMMARY & NEXT STEPS

In this white paper, we have explored NVIDIA's NVSMI tool (nvidia-smi) in depth – from installation and basic usage to advanced monitoring and control features – all in the context of WOLF's mission-critical GPU hardware. NVSMI serves as an **indispensable tool for internal engineers, sales engineers, product teams, and customer support** when dealing with GPU-powered modules like the WOLF-3476 XMC and other WOLF products. It enables us to monitor real-time GPU metrics, investigate and resolve performance bottlenecks, manage power and thermal constraints, and assure the reliability of GPUs operating in harsh environments.

KEY TAKEAWAYS:

- NVSMI provides **immediate visibility** into GPU status (utilization, temperature, clocks, power, etc.) which is crucial for debugging and optimizing system performance.
- Using NVSMI, we can **fine-tune GPU behavior** (through power limits and clock settings) to meet the unique SWaP challenges of aerospace and defense applications without hardware modifications.
- The tool helps bridge the gap between expected performance and actual behavior by offering data that explains why a GPU might be throttling or underutilized, thus guiding solutions (better cooling, code optimization, load balancing, etc.).
- NVSMI's outputs (text or CSV) can be integrated into scripts and support workflows, making it a **practical choice for both development and field maintenance** of WOLF products.

NEXT STEPS – BEYOND NVSMI:

This training focused on interactive command-line usage of NVSMI. As we advance, the next phase is to look into **NVML (NVIDIA Management Library)** and possibly NVIDIA's System Management SDK for programmatic control. NVML is the underlying API that NVSMI uses. By using NVML in C/C++ or its Python bindings, we could create custom monitoring daemons or integrate GPU telemetry into WOLF's system software directly. For example, a WOLF high-performance computer could have a health monitor service that calls NVML functions to log GPU stats or trigger alerts if thresholds are crossed. NVML would allow more fine-grained control and continuous integration of GPU monitoring, beyond the on-demand snapshots NVSMI provides.

CONCLUSION:

NVSMI is a powerful tool for unlocking the full potential of NVIDIA GPUs in rugged, mission-critical systems. By mastering its installation, usage, and output interpretation, engineers can better design, optimize, and support GPU-based solutions, ensuring reliable performance in demanding environments like aerospace and defense. This white paper, along with hands-on examples using the WOLF-3476, offers a strong starting point. The next step is to apply these commands in real-world scenarios and share findings across the team. With this knowledge, even newer engineers can become effective troubleshooters and contribute to maintaining high system performance in the field.

WOLF
ADVANCED TECHNOLOGY