

Theory: GPU Acceleration & Performance

A comprehensive guide to leveraging GPU computing power for efficient hash cracking with Hashcat.

Why GPUs Excel at Hash Cracking

Hash functions like MD5, SHA-1, and bcrypt are designed to be computed independently for each input. There are no data dependencies between one hash computation and the next. This makes hashing an **embarrassingly parallel** workload — the ideal scenario for GPU computing.

A modern CPU has 8–24 cores, each capable of complex branching and sequential logic. A modern GPU has thousands of simpler cores (NVIDIA calls them CUDA cores; AMD calls them stream processors) designed to execute the same operation on many data points simultaneously. When cracking passwords, each GPU core can compute a hash for a different password candidate at the same time.

The result is dramatic. A single NVIDIA RTX 4090 can compute approximately 160 billion MD5 hashes per second, while a top-tier Intel or AMD CPU manages roughly 5 billion. For slower hash algorithms like bcrypt, the gap narrows but GPUs still hold a significant advantage.

CPU vs GPU Architecture

Property	CPU	GPU
Core count	8–24 (high-end desktop)	5,000–16,000+ (modern GPUs)
Clock speed	3–6 GHz per core	1.5–2.5 GHz per core
Strength	Complex sequential tasks	Massively parallel simple tasks
Memory	System RAM (32–128 GB typical)	VRAM (8–24 GB typical)
Best for	General computing, OS tasks	Hash cracking, rendering, ML

System Preparation

GPU acceleration in Hashcat requires the correct drivers and compute frameworks. Without them, Hashcat cannot detect or use the GPU.

NVIDIA GPUs

1. **Install the latest NVIDIA driver** from the [NVIDIA driver page](#). On Linux, use your distribution's package manager or the official `.run` installer.
2. **Install the CUDA Toolkit** from the [NVIDIA CUDA page](#). CUDA includes OpenCL support, so you get both backends.
3. **Verify the installation:**

```
# Check NVIDIA driver and GPU status
nvidia-smi

# Expected output includes GPU name, driver version, CUDA version,
# temperature, power usage, and memory usage
```

AMD GPUs

1. **Install AMD drivers:** On Windows, install AMD Adrenalin Edition. On Linux, install ROCm or the AMDGPU-PRO driver with OpenCL support.
2. **Verify OpenCL availability:**

```
# Check OpenCL platforms and devices
clinfo

# Look for your GPU listed under a platform
# Ensure "Device Type" shows "GPU"
```

Backend support: Hashcat supports three compute backends: **CUDA** (NVIDIA only, often fastest), **OpenCL** (both vendors), and **HIP** (AMD ROCm). If you have an NVIDIA GPU, CUDA is recommended. For AMD, OpenCL or HIP depending on your driver setup.

Detecting Devices with `hashcat -I`

Before running any attack, always verify that Hashcat sees your GPU. The `-I` flag lists all detected compute devices:

```
hashcat -I
```

The output shows each platform (CUDA, OpenCL) and its devices with their IDs, names, memory, and capabilities. Pay attention to:

- **Device ID** — used with `-d` to select specific GPUs
- **Device Type** — GPU vs CPU
- **Memory** — larger VRAM allows bigger wordlists and more complex attacks
- **Driver Version** — outdated drivers can cause crashes or poor performance

Device Selection

Hashcat provides two flags for controlling which hardware is used:

Device Type (`-D`)

Selects the category of device:

- `-D 1` — CPU only
- `-D 2` — GPU only
- `-D 1,2` — Both CPU and GPU

In almost all cases, `-D 2` is the correct choice. Using both CPU and GPU simultaneously can actually slow things down because the CPU processes hashes much slower and creates synchronization overhead.

Specific Devices (`-d`)

When you have multiple GPUs, use `-d` to select which ones to use:

```
# Use only GPU #1
hashcat -m 0 -a 0 -D 2 -d 1 hashes.txt rockyou.txt

# Use GPUs #1 and #2
hashcat -m 0 -a 0 -D 2 -d 1,2 hashes.txt rockyou.txt
```

```
# Use all available GPUs (default when -D 2 is set)
hashcat -m 0 -a 0 -D 2 hashes.txt rockyou.txt
```

Tip: If your primary GPU also drives the display, running Hashcat at full load on it can freeze or crash your display. Consider using `-d 2` to target a secondary GPU, or lower the workload profile.

Workload Profiles

The `-w` flag controls how much GPU time Hashcat demands. Higher values mean faster cracking but more system impact.

Profile	Name	Behaviour	Recommended Use
<code>-w 1</code>	Low	Short GPU compute bursts with frequent pauses. Desktop remains fully responsive.	Cracking while working on the same machine
<code>-w 2</code>	Medium	Default balance between speed and responsiveness.	General-purpose cracking
<code>-w 3</code>	High	Long GPU compute bursts. System may lag noticeably.	Dedicated cracking sessions
<code>-w 4</code>	Maximum	GPU is fully saturated. Display may freeze.	Headless servers or secondary GPUs

The difference between `-w 1` and `-w 4` can be 2–5x in speed depending on the hash type and GPU. For fast hash types like MD5, the impact is smaller because the GPU is already fast. For slow hashes like bcrypt, the profile makes a bigger difference.

Temperature Management

Sustained GPU cracking generates significant heat. Without thermal management, GPUs can throttle (reducing performance) or shut down to prevent hardware damage. Hashcat provides two temperature controls:

`--gpu-temp-abort=N`

If any GPU reaches N degrees Celsius, Hashcat immediately stops the session and saves progress. This is a safety net to prevent hardware damage. A reasonable value is 85–90 degrees, depending on your GPU model's thermal limits.

`--gpu-temp-retain=N`

Hashcat dynamically adjusts the workload to keep the GPU at or below N degrees. When the temperature approaches the limit, Hashcat reduces the compute load. When it drops, Hashcat increases it again. Set this 10–15 degrees below the abort temperature.

```
# Recommended thermal configuration
hashcat -m 0 -a 0 -D 2 -w 3 \
  --gpu-temp-abort=85 \
  --gpu-temp-retain=75 \
  hashes.txt rockyou.txt
```

Cooling matters: Ensure adequate case airflow. Clean dust from GPU heatsinks regularly. In multi-GPU rigs, consider liquid cooling or leaving PCIe slot gaps between GPUs for better airflow. An overheating GPU will thermal-throttle and actually be slower than a properly cooled GPU at a lower workload profile.

Advanced Performance Tuning

For most users, `-w 3` or `-w 4` is sufficient. Advanced users can fine-tune kernel parameters for specific GPU architectures:

Kernel Acceleration (`-n`)

Controls the number of threads per compute block. Higher values increase parallelism but also increase GPU memory usage. This is most relevant for NVIDIA GPUs. Hashcat's auto-tuner usually picks a good value, but for some hash modes you may benefit from manual adjustment.

Kernel Loops (`-u`)

Controls the number of iterations per work unit. Larger values reduce overhead from kernel launches but increase the time per unit. This is more relevant for AMD GPUs and for slow hash types where each iteration is expensive.

Kernel Threads (`-T`)

Sets the vector width (number of elements processed per thread). Usually 1, 2, 4, 8, or 16. Auto-tuning handles this well for most cases.

Best practice: Run Hashcat with `-w 3` and no manual kernel flags first. Note the speed. Then experiment with `-n` and `-u` values, comparing speed each time. The Hashcat benchmark (`hashcat -b`) auto-tunes these for each hash mode.

Runtime Controls

For scheduled or time-limited cracking sessions, Hashcat supports:

- `--runtime=N` — Stop after N seconds. The session is saved automatically and can be resumed with `--restore`.
- `--status-timer=N` — Print status information every N seconds. Useful for monitoring long-running sessions.

```
# Run for 1 hour, status every 60 seconds
hashcat -m 0 -a 3 -D 2 -w 3 \
  --runtime=3600 \
  --status-timer=60 \
  hashes.txt ?a?a?a?a?a?a

# Resume a previously stopped session
hashcat --restore
```

Benchmarking

Hashcat includes a built-in benchmark mode that tests performance across all supported hash types on your hardware:

```
# Benchmark all hash types
hashcat -b

# Benchmark a specific hash type
hashcat -b -m 0      # MD5
hashcat -b -m 1000   # NTLM
hashcat -b -m 3200   # bcrypt
```

Benchmark results help you understand your hardware's capabilities and set realistic expectations for cracking times. They also help verify that your GPU is performing correctly — if benchmark speeds are significantly lower than expected for your GPU model, there may be a driver or thermal issue.

Troubleshooting Common Issues

Low Hash Speed

- Verify you are using `-D 2` (GPU mode), not defaulting to CPU
- Increase the workload profile to `-w 3` or `-w 4`
- Update GPU drivers to the latest stable version
- Check for thermal throttling with `nvidia-smi` or `sensors`
- Close other GPU-intensive applications (games, video editors, browsers with hardware acceleration)

GPU Overheating

- Set `--gpu-temp-retain` to keep temperatures in check
- Reduce the workload profile (`-w 2` or `-w 1`)
- Improve physical cooling: clean dust, add case fans, improve cable management
- Increase GPU fan speed using `nvidia-settings` or MSI Afterburner

Multi-GPU Issues

- Ensure all GPUs have the same driver version
- Check PCIe lane allocation: x16 vs x8 vs x4 affects bandwidth
- Use `hashcat -I` to verify all GPUs are detected
- Test each GPU individually with `-d 1`, `-d 2`, etc., to isolate problems

Driver Problems

- Completely uninstall old drivers before installing new ones (use DDU on Windows)
- On Linux, ensure the correct kernel module is loaded: `lsmod | grep nvidia`
- Check for OpenCL/CUDA version compatibility with your Hashcat version
- Some laptop GPUs (Optimus/hybrid graphics) require special configuration

RAM / VRAM Limitations

- Large wordlists can exceed GPU memory. Hashcat handles this by streaming, but performance may drop.
- Some hash modes with large salt counts require significant VRAM. Check Hashcat's output for memory warnings.
- If you see "out of memory" errors, reduce the number of hashes being cracked simultaneously, or use a GPU with more VRAM.

Further Reading

- [Hashcat Official Wiki](#)
- [Hashcat Benchmark Results](#)
- [Hashcat Hardware Monitoring FAQ](#)
- [NVIDIA CUDA Toolkit](#)
- [AMD ROCm Documentation](#)