GPU SMITH
COMPUTE ENGINEERING

GPU SMITH / RESEARCH & PRACTICAL GUIDANCE

CUDA to ROCm Porting: HIPIFY Coverage and Manual Rework

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

Published by GPU Smith · 2026-10-13 · cuda to rocm porting · hipify · hipify clang · hipify perl · hip · rocm · gpu migration · amd instinct · cuda libraries

Original article: https://gpusmith.com/articles/en/cuda-to-rocm-porting-hipify-coverage

In this article

1. Executive Summary
2. Introduction and Background
3. Key Changes: From Translation to Migration Scope
4. Implementation Considerations and Process Changes
5. Dependency Inventory and Remediation Classes
6. Validation, Performance Tuning, and Operations
7. Data Analysis and Evidence
8. Implications and Future Directions
9. Conclusion

Executive Summary

CUDA to ROCm porting is a source-conversion and application-validation project, not a compatibility percentage. **HIPIFY** can translate supported CUDA constructs into the **Heterogeneous-compute Interface for Portability (HIP)**, but its own documentation explicitly limits automatic conversion. A translated application still needs dependency, build, numerical, performance, and deployment evidence. This report recommends a repository inventory with **six remediation classes**: auto-translated, translated with warnings, manually portable, substitute library/API, redesign, and currently blocked. These are proposed planning categories, not vendor certification levels. (rocm.docs.amd.com) (github.com)

As accessed on **October 13, 2026**, AMD's latest release notes describe **ROCm Core SDK 10.1.0**, released **October 5, 2026**. (rocm.docs.amd.com) That release name does not establish that every framework package or installed HIPIFY executable matches it. JAX requires its ROCm plugin's target version to match the installed ROCm. (docs.jax.dev) ONNX Runtime documents removal of its ROCm Execution Provider beginning with **release 1.23**, directing users toward MIGraphX. (onnxruntime.ai) Buyers should pin the translator, runtime, framework, libraries, compiler, operating system, and GPU as separate evidence items.

The strongest candidates for predominantly mechanical translation have accessible source, supported API calls, reproducible compiler inputs, and dependencies with verified replacement paths. CMake can export exact translation-unit compiler calls, providing a stronger inventory basis than counting checked-in CUDA files alone. (cmake.org) Conversely, inline Parallel Thread Execution (PTX), generated extensions, and unavailable dependency packages deserve individual review. NVIDIA documents inline PTX as arbitrary PTX inserted into CUDA programs. (docs.nvidia.com) PyTorch supports just-in-time extensions from source strings, which can escape a static file scan. (docs.pytorch.org)

The procurement deliverable should therefore contain a blocker register, call-level library map, reproducible build evidence, correctness tolerances, representative performance results, and a rollback path. Performance claims need the same workload and accuracy conditions: MLCommons requires accuracy validation for each distinct Endpoints configuration. (mlcommons.org) It defines concurrency as simultaneous queries in flight. (mlcommons.org) No buyer repository or AMD GPU was supplied for this report, so it reports **no measured application compatibility rate, migration duration, or performance parity**. The worksheet below provides the evidence structure needed to obtain those results before committing to hardware.

Introduction and Background

The useful go/no-go question is whether a particular application can deliver its required behavior on a specified AMD system with an acceptable remediation scope. Asking whether “CUDA works on ROCm” omits the source files, binary dependencies, numerical modes, framework packages, and operating environment that determine the answer. The inventory should distinguish accessible source from binary dependencies whose required implementation has not been validated on the selected AMD system. Establish that distinction before interpreting a hardware benchmark as a procurement decision.

This report takes an independent engineering-advisor perspective. **GPU Smith** describes its infrastructure work in terms of written acceptance criteria. (gpusmith.com) That is the relevant first-party perspective here, rather than a competing accelerator or cloud offering. The proposed migration packet uses acceptance

evidence to connect repository findings with the purchasing decision. It does not infer an AMD implementation track record, partnership, or performance result from the consultancy's website.

ROCm is treated here as a versioned software stack, while HIP is the programming interface and HIPIFY the translation tooling. Framework packaging remains a separate layer: JAX distinguishes cross-platform Python code from compiled accelerator-specific binaries. A Python application can therefore have little visible CUDA syntax yet still depend on a particular binary distribution. (docs.jax.dev)

The report addresses the migration workflow, supported-API evidence, kernel review, library substitutes, extension builds, and operational qualification. It does not compare accelerator vendors generally or provide an upgrade runbook for an existing ROCm installation. The primary sources were opened during this research session. Their release labels are reported individually because a “latest” documentation page is not a lockfile. Reproducible Builds identifies dependency versions, flags, and environment variables as parts of the build environment; a migration record should preserve those inputs explicitly. (reproducible-builds.org)

Key Changes: From Translation to Migration Scope

A source translation is an intermediate artifact

HIPIFY coverage answers a narrower question than application compatibility: which constructs the selected translator recognizes and maps. Its supported-API index includes runtime, driver, device, runtime-compilation, and library families. That index is useful for discovery, but its family names do not establish that every call, option, or behavior in an application is supported. Generate the installed translator's mapping documentation and compare actual call sites against it. (rocm.docs.amd.com)

Use **five migration layers** as an analyst-defined checklist:

- **Syntax and APIs:** Identify calls, launch expressions, intrinsics, and embedded device code.
- **Dependencies:** Identify libraries, frameworks, plugins, binary artifacts, and their replacement paths.
- **Build and toolchain:** Capture compiler inputs, target architectures, linking, and extension generation.
- **Numerical and performance behavior:** Define acceptable results and representative workload measurements.
- **Deployment and operations:** Reproduce packaging, scheduling, observability, and offline installation.

These layers have different acceptance tests. A source diff belongs to the first layer. A dependency package that installs and exercises the needed operations belongs to the second. Neither should close a numerical or operational finding without evidence.

hipify-clang versus hipify-perl

hipify-clang parses an abstract syntax tree and applies transformation matchers. Its documentation requires appropriate CUDA inputs, include paths, and definitions. **hipify-perl** instead relies heavily on regular expressions and has no dependency on CUDA or other third-party tools. That makes it a useful reconnaissance option when the full parsing environment is unavailable, while making its successful output a weaker basis for accepting complex source transformations. (rocm.docs.amd.com) (rocm.docs.amd.com)

The Perl documentation lists limitations involving host/device call differentiation and other complex constructs, and recommends hipify-clang for advanced translation needs. Preserve the choice of tool in every inventory record. A file examined only by regular-expression replacement should not inherit the confidence level of a file parsed under its actual compilation settings. (rocm.docs.amd.com) (rocm.docs.amd.com)

Kernel portability requires semantic review

Inventory inline PTX separately from ordinary CUDA calls. NVIDIA's documentation explains that the compiler front end does not understand the contents of the assembly template string. Accordingly, replacing surrounding API names is insufficient evidence that the embedded instruction sequence has an AMD implementation. Classify such findings as manually portable or redesign until an equivalent operation and its acceptance tests are established. This is an engineering inference from the documented assembly mechanism. (docs.nvidia.com)

Audit subgroup assumptions as well. AMD's porting guide says code should not assume a warp size of **32 or 64**, and emphasizes the correct lane-mask type. Review reductions, ballots, shuffles, indexing, and synchronization wherever their correctness relies on a fixed group width. An inventory hit should record the assumption and test requirement, rather than automatically labeling the whole kernel unsupported. (rocm.docs.amd.com)

The runtime model also changes

AMD documents separate CUDA driver and runtime APIs, while HIP mostly uses a single API. Driver-context creation, module loading, runtime compilation, and process lifecycle code therefore need explicit review beyond renamed symbols. Identify which application component owns those behaviors and whether a third-party plugin implements them invisibly. (rocm.docs.amd.com)

Implementation Considerations and Process Changes

Preserve inputs before running HIPIFY

Create an isolated migration branch and capture the original successful build. Use a compilation database where supported: CMake's export option records exact compiler calls, but is implemented by its Makefile and Ninja generators. Retain configuration evidence for projects using other generators instead of assuming an absent database means there are no CUDA translation units. (cmake.org) (cmake.org)

Hypothetical Example: The following shell commands are **documentation-based examples**, not commands executed against a buyer repository in this report. Run them inside a separate migration checkout. The evidence directory is an output location chosen by the maintainer, and the CMake generator must be available in that environment.

```
mkdir -p migration-evidence
git rev-parse HEAD > migration-evidence/source-commit.txt
git status --porcelain > migration-evidence/source-state.txt
cmake -S . -B build-cuda -G Ninja -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
cp build-cuda/compile_commands.json migration-evidence/
```

```
hipify-clang --version > migration-evidence/hipify-version.txt
hipify-clang --versions > migration-evidence/hipify-dependencies.txt
hipify-clang --help > migration-evidence/hipify-help.txt
hipify-clang --examine -p build-cuda src/kernel.cu \
  > migration-evidence/kernel.stats.txt \
  2> migration-evidence/kernel.diagnostics.txt
hipify_exit_code=$?
printf '%s\n' "$hipify_exit_code" > migration-evidence/kernel.exit-code.txt
hipify-clang --csv --doc-format=full --doc-roc=joint \
  -o migration-evidence
```

Replace the illustrative source path with a file listed in the captured compilation database and review its compilation settings before inspection. The `-p build-cuda` option selects that database; omit the `--` separator in database mode. (AMD HIPIFY usage guide) HIPIFY documents **--examine** as combining inspection without output and statistics reporting, and **--versions** as reporting supported third-party software versions. The supported-API index documents CSV mapping generation. Capture command exit codes too; redirecting an error into an evidence file does not make the inspection successful. (rocm.docs.amd.com) (rocm.docs.amd.com)

Maintain an explicit coverage denominator

Use a manifest of compilation units and separately list generated code, vendored code, dynamically constructed extensions, and binary-only dependencies. PyTorch's source-string extension loader is a concrete reason a checked-in file inventory can be incomplete. A binary artifact has no meaningful “translated lines” denominator; it needs a dependency-level disposition. (docs.pytorch.org)

For each item, preserve:

- **Identity:** Repository commit, file or module, and dependency version.
- **Build context:** Compiler invocation, definitions, include paths, and enabled configuration.
- **Scan result:** Tool version, command, exit code, diagnostics, and translated output.
- **Coverage exception:** Generated, excluded, unavailable, or binary-only material.
- **Critical path:** Whether the item runs in representative acceptance workloads.
- **Disposition:** Proposed remediation class, evidence needed, and assigned owner.
- **Retest trigger:** Which source, compiler, library, or package changes reopen the finding.
- **Closure:** Review result, successful tests, and links to retained artifacts.

HIPIFY provides per-file and aggregate translation statistics. Keep those raw outputs, then reconcile them against the manifest. Report “successfully examined units divided by units requiring examination” alongside excluded units, parse failures, and unreviewed dependencies. This report deliberately does not turn that ratio into a probability that the application will run. (rocm.docs.amd.com)

Keep **API occurrence counts** separate from file coverage and remediation counts. Record recognized calls, emitted replacements, unsupported-call diagnostics, unknown identifiers, and parse failures under documented counting rules. Manually sample critical paths to check that the logs correspond to the code actually built. A warning count is not automatically a count of distinct unsupported API names, and an unparsed file belongs in the missing-evidence register rather than the successful-translation total.

Separate translation from the target build

For CMake-based HIP builds, explicitly capture **HIP_ARCHITECTURES** and the configured target list. CMake documents whole and separable compilation modes, and its architecture default can depend on detected hardware or the compiler. A build performed on a developer workstation should not silently determine the procurement target. (cmake.org) (cmake.org) (cmake.org)

Review compiler options rather than copying CUDA flags unchanged. AMD's PyTorch-extension guidance specifically calls for removing nvcc-specific options unsupported by hipcc. LLVM documents processor-specific features and warns that feature mismatches can affect execution. LLVM documents target identifiers supplied through **mcpu** or **offload-arch** compiler options. Retain both compiler arguments and the resulting device-code targets as acceptance evidence. (github.com) (llvm.org) (llvm.org)

Dependency Inventory and Remediation Classes

Map operations, not just library names

Table 1 lists documented candidate replacements. “Candidate” means an interface or implementation to evaluate against the application's actual operations. It does not assert identical algorithms, package availability, numerical results, licensing terms, or performance.

CUDA dependency	Documented candidate	Manual review and acceptance evidence
cuBLAS	**hipBLAS**, with rocBLAS-oriented backend dependencies. Its interface is documented as compatible with rocBLAS and cuBLAS-v2. (github.com)	Inventory the exact routine, datatype, layout, workspace, stream, and selected algorithm. Retain result tolerances and representative timings.
cuFFT	**hipFFT**, with rocFFT or cuFFT backends. (github.com)	Test plan creation, transforms, workspace, normalization, and application-specific callbacks or multi-device behavior.
cuSPARSE	**hipSPARSE**, documented as compatible with rocSPARSE and cuSPARSE-v2 interfaces. (github.com)	Validate the actual sparse format, indexing, precision, and required routines.
cuRAND	**hipRAND**, documented as compatible with rocRAND and cuRAND-v2 interfaces. (github.com)	Define whether acceptance requires matching distributions, statistical properties, or identical sequences.
CUB	**hipCUB**, using rocPRIM in the ROCm environment. (github.com)	Check device-wide and subgroup algorithms, temporary storage, and ordering expectations.
cuDNN	**MIOpen**, AMD's machine-learning primitives library. (github.com)	Review operations, tensor descriptors, supported layouts, workspace, and algorithm-selection lifecycle.
NCCL	**RCCL** as a candidate collective implementation; its documented operations include all-reduce, all-gather, and reduce-scatter. (github.com)	Verify every used collective and datatype, process initialization, topology, transport, and performance. No drop-in parity is assumed.

The table separates interface proximity from application acceptance. For example, hipBLAS itself has underlying dependencies, while MIOpen's porting guide documents differences from cuDNN and a convolution sequence involving workspace inquiry, algorithm finding, and execution. A mapping row should therefore point to tests for the actual lifecycle used by the application. (github.com) (rocm.docs.amd.com)

Record dependency licenses and redistribution conditions from the exact source release and binary distribution selected. This report does not infer a common license for the whole ROCm ecosystem. Preserve license files and provenance in the packaging record, and seek qualified licensing advice when the intended redistribution terms need interpretation.

Classify remediation without estimating labor

Table 2 defines the proposed classifier. Its examples are labeled **Hypothetical Example**, not observed outcomes from an executed migration. Assign the class at the finding level; one module can contain multiple findings with different closure conditions.

Remediation class	Decision rule	Example acceptance evidence
Auto-translated	Recognized source transformation produces a reviewed candidate without relevant diagnostics.	Retained source diff, successful target build, and passing tests. Translation alone does not close the item.
Translated with warnings	A candidate is emitted, but diagnostics or experimental mappings remain.	Warning disposition, pinned mapping evidence, and targeted correctness tests.
Manually portable	Source is available and a reviewed HIP implementation can preserve the required behavior.	Reviewed patch and edge-case tests for the identified assumption.
Substitute library/API	The application needs a different documented dependency or interface.	Required operations demonstrated on the pinned dependency and target stack.
Redesign	Required behavior needs a different algorithm, execution model, or component boundary.	Accepted design, reference results, integration tests, and representative performance evidence.
Currently blocked	No validated path exists within the selected requirements and available evidence.	Specific unmet requirement, supporting evidence, owner, and condition that would reopen evaluation.

These classes do not imply increasing migration days. A small redesign can be bounded; a seemingly simple substitution can be critical because it affects every output. Track experimental mappings and diagnostic suppression in the command record. A chosen setting changes what the inspection reports; it should not erase the review obligation.

Keep a repository CSV

Hypothetical Example: The following **CSV inventory template** contains illustrative paths only. Blank evidence and owner fields remain unresolved. Status and remediation class are distinct: a finding can remain open after its candidate class is known.

```
file_module,cuda_dependency,call_library,hipify_outcome,proposed_equivalent,manual_review,criticality,owner,
evidence,status,remediation_class
```

```
src/example.cu,CUDA runtime,allocation and copy,not run,HIP,required,unassessed,,,open,unclassified
extension/generated,generated source,custom kernel,not captured,HIP
candidate,required,unassessed,,,open,unclassified
vendor/binary,binary dependency,library entry point,not
scannable,unverified,required,unassessed,,,open,currently blocked
```

Add separate fields for build configuration, dependency release, target hardware, and license evidence when needed. The binary row is blocked pending evidence within this template, not a factual claim that its hypothetical library lacks an AMD implementation. Generated extension code should be captured during build or runtime discovery because PyTorch supports just-in-time compilation from strings. (docs.pytorch.org)

Framework names do not settle the inventory

PyTorch intentionally reuses torch.cuda interfaces in its HIP implementation. Searching for that namespace alone overstates CUDA-only dependency. Its AMD-maintained extension wiki describes CUDAExtension hipification, but this is workflow evidence, not proof that every current extension builds. Toolchains also require separate installation beyond torch. (docs.pytorch.org) (github.com) (docs.pytorch.org)

TensorFlow exposes a ROCm configuration prompt in its source-build documentation and requires **Bazel**. Those facts establish a possible build path, not a validated package/GPU combination. **CuPy** describes ROCm support as experimental and lists unavailable functionality including multi-GPU FFT and FFT callbacks. CuPy additionally documents unavailable recent upstream ROCm wheels and images, so packaging needs separate validation. JAX requires matching the plugin's ROCm target version. Preserve framework-specific requirements as separate dependency findings. (www.tensorflow.org) (www.tensorflow.org) (docs.cupy.dev) (docs.cupy.dev) (docs.cupy.dev) (docs.jax.dev)

Validation, Performance Tuning, and Operations

Correctness precedes performance interpretation

Use the established CUDA application and an appropriate independent reference as separate comparators. Agreement with one existing output is useful evidence, but should not replace a declared acceptance requirement. PyTorch documents hardware numerical differences for **TensorFloat-32 (TF32)** on AMD Instinct MI300. (docs.pytorch.org) Record precision modes rather than assume identically named options imply identical arithmetic.

The following validation categories are proposed test requirements:

- **Functional results:** Check output shapes, values, boundary cases, and error handling.
- **Tolerance:** Define absolute and relative error, affected operations, and reference precision.
- **Determinism:** Decide which runs require repeatability and distinguish it from cross-platform identity.
- **Memory behavior:** Exercise allocation pressure, buffer reuse, leaks, and invalid accesses.
- **Streams and synchronization:** Test dependencies, concurrency, and asynchronous completion.
- **Collectives:** Validate results, initialization, process counts, and representative message sizes.
- **Failure handling:** Exercise compilation errors, missing packages, startup failures, and device loss.
- **Regression coverage:** Retain tests for every manual patch and every warning disposition.

Triton's documentation points AMD debugging toward LLVM AddressSanitizer, while its interpreter does not cover indirect memory-access patterns. A host-side or interpreter test should therefore complement device execution rather than certify every kernel. The Triton repository also documents its address-sanitizer build option as currently supported only on the AMD backend. Preserve the debugging configuration with the findings it supports. (triton-lang.org) (triton-lang.org) (github.com)

Tune a representative workload

Measure the application at its relevant operating point before optimizing isolated kernels. MLPerf's submission guidance distinguishes scenarios according to real-world usage and requires outputs to satisfy accuracy thresholds. These principles support a migration plan that fixes workload shape and correctness requirements before comparing speed. They do not supply a CUDA-to-ROCm benchmark result. (docs.mlcommons.org) (docs.mlcommons.org)

Record batch size, precision, shapes, memory consumption, latency distribution, throughput, startup time, and compilation overhead where relevant. Capture CPU placement and memory-node policy as well: Docker documents its memory-node constraint as effective only on non-uniform memory access (NUMA) systems. (docs.docker.com) MLCommons Endpoints explicitly characterizes complete-system performance and defines concurrency as simultaneous queries in flight. An infrastructure buyer should therefore distinguish a kernel timing from a service measurement under a stated concurrency target. (mlcommons.org) (mlcommons.org)

Only then examine launch configuration, work partitioning, layout, fusion, library algorithm selection, and communication overlap. Target-feature choices must match the selected processor, according to LLVM. MIGraphX also exposes exhaustive tuning during compilation, illustrating why startup and compile costs belong beside steady-state results. Report any tuning configuration and its cost rather than publishing an unexplained “optimized” number. (llvm.org) (onnxruntime.ai)

Validate communication and the operating envelope

RCCL's documented transports include InfiniBand Verbs and TCP/IP sockets. Applications using Message Passing Interface (MPI) need a separate transport check: Open MPI's **v5.0.x** documentation describes ROCm-aware access to AMD device buffers and formally supports **Unified Communication X (UCX)** for that path. A successful collective-library test does not establish the MPI build's capabilities. Open MPI documents **ompi_info** for checking ROCm build support; retain that output. (github.com) (docs.open-mpi.org) (docs.open-mpi.org) (docs.open-mpi.org)

Docker supports selected host-device access, supplementary groups, and image digests. Preserve the digest, device exposure, and permissions in deployment evidence. Linux cgroup device controls govern access to device files, so an executable that runs interactively can still fail within a different container policy. Linux's cgroup v2 device controller is implemented through cgroup Berkeley Packet Filter (BPF) programs, rather than controller interface files. Avoid treating a packaging problem as a kernel portability result. (docs.docker.com) (docs.docker.com) (docs.docker.com) (docs.kernel.org) (docs.kernel.org)

Kubernetes' generic device-plugin documentation describes integer, non-overcommittable resources and states that devices cannot be shared between containers in that allocation model. If a deployment requires partitioning or sharing, qualify the chosen implementation explicitly. Monitoring agents can be deployed as a

daemon or DaemonSet; retaining that deployment and verifying its metrics is part of operational acceptance. (kubernetes.io) (kubernetes.io) (kubernetes.io)

Data Analysis and Evidence

What the available numbers establish

Table 3 records source-specific anchors and their limits. These are documentation facts observed on the research access date, not a single interoperable software bill of materials. Versioned pages and unversioned pages are identified separately.

Evidence item	Verified anchor	Planning interpretation
Current AMD release notes	**ROCm Core SDK 10.1.0**, dated **October 5, 2026**. (rocm.docs.amd.com)	Capture the deployed runtime independently from the current documentation label.
HIPIFY tool dependencies	The current usage page lists **CUDA 13.2.0** and recommends stable **Clang 22.1.8**. (rocm.docs.amd.com)	These are translator dependencies, not a guarantee for every runtime and framework package.
CMake HIP architecture control	HIP_ARCHITECTURES was added in **CMake 3.21**, as documented on its versioned **3.30** page. (cmake.org)	Verify the project's actual CMake version and target properties.
ONNX Runtime provider path	The ROCm Execution Provider is documented as removed since **1.23**. (onnxruntime.ai)	Evaluate MIGraphX and the application's required graph operations.
CuPy packaging	The installation page says upstream CuPy ROCm wheels and Docker images are unavailable starting with **v13.4.0**; it separately identifies AMD-hosted **ROCm 6.4** wheels distributed as **amd-cupy**. (docs.cupy.dev)	Validate the specific amd-cupy distribution with its ROCm 6.4 runtime, target GPU, and required operations before acceptance, or qualify a source build. Treat ROCm 10.1.0 as a separate combination requiring its own validation.
GPU/OS example	The current ROCm matrix lists **Ubuntu 24.04.5**, **GA kernel 6.8**, for the **AMD Instinct MI350 Series**. (rocm.docs.amd.com)	One documented combination is not evidence for another GPU, kernel, or OS.
MLCommons Endpoints method	A submission contains **7 to 32 points**. (mlcommons.org)	This is benchmark-methodology evidence, not the number of tests sufficient for a migration.

The key quantitative conclusion is a limit: these numbers describe releases, support entries, and testing methodology. They provide **no application-specific conversion percentage or speed ratio**. JAX's plugin-version matching requirement reinforces why an attractive current runtime release cannot automatically close every framework inventory item. (docs.jax.dev)

Count complexity without disguising omissions

Define the countable object before calculating anything. For this worksheet, use a **finding** as one dependency or behavior requiring a distinct remediation decision. Record its class once, even when several files share it. Maintain file coverage separately so consolidating a shared library finding does not conceal unexamined

source.

Let **T** be the translation units requiring examination and **E** the units successfully examined under their required compile settings. Report examination coverage as **E / T**, with the manifest defining both sets. Let **N** be all inventory findings and report the count in each remediation class, plus the unclassified count. These are proposed accounting definitions, not measured values. Reconcile the declared sets with retained inspection reports and compiler-call evidence. CMake's export provides the latter. (cmake.org)

Use the following Python example on an actual completed CSV. It reports class counts and missing evidence, not estimated labor or compatibility.

```
import csv
from collections import Counter

with open("repository-inventory.csv", newline="", encoding="utf-8") as stream:
    rows = list(csv.DictReader(stream))

classes = Counter(row["remediation_class"] or "unclassified" for row in rows)
print("inventory_rows", len(rows))
for name, count in sorted(classes.items()):
    print(name, count)
print("rows_without_evidence", sum(not row["evidence"].strip() for row in rows))
print("rows_without_owner", sum(not row["owner"].strip() for row in rows))
```

No values from this script are claimed here because no buyer CSV was supplied. A count by class should always accompany criticality and ownership; otherwise numerous low-impact translations can obscure a single unresolved production dependency.

Keep build and performance evidence distinct

Reproducible Builds defines artifact reproducibility in terms of recreating identical artifacts and verifying them by bitwise comparison. That is separate from obtaining numerically acceptable GPU results. Likewise, a Spack manifest can resolve differently under different configuration, while a compatible-machine lockfile preserves initial concrete specifications. Retain the package lock, image digest, source commit, build commands, and test record as separate evidence objects. (reproducible-builds.org) (reproducible-builds.org) (spack.readthedocs.io) (spack.readthedocs.io)

The report has not executed HIPIFY against a public sample or run a device benchmark. Documented commands and examples demonstrate the inventory method only. Procurement-grade evidence must come from the buyer's relevant application paths and selected operating environment.

Implications and Future Directions

Turn the inventory into a procurement packet

A procurement team should receive an explicit qualification boundary before requesting a final hardware commitment. AMD's compatibility matrix describes a coordinated firmware, driver, and user-space stack. The packet should therefore specify the complete supported combination rather than simply naming “ROCm compatible” hardware. Archive the matrix entry used and record the date it was checked. (rocm.docs.amd.com)

Include the following proposed packet contents:

- **Scope boundary:** Repository commits, configurations, representative workloads, and excluded material.
- **Blocker register:** Required behaviors without a validated path and the condition for reconsideration.
- **Dependency map:** Exact calls, candidate libraries, package sources, and redistribution evidence.
- **Build record:** Compiler and translator identities, target architectures, flags, and retained diagnostics.
- **Support matrix:** GPU model, firmware, driver, OS, kernel, runtime, and framework versions.
- **Correctness record:** References, tolerances, precision settings, deterministic requirements, and failures.
- **Performance record:** Workload, concurrency, accuracy conditions, run configuration, and retained measurements.
- **Operating plan:** Containers, scheduling, monitoring, offline installation, recovery, and rollback.

Ask vendors to identify the exact supported package combination and demonstrate the required operations. Retain answers to these questions:

- **Package support:** Which exact framework, runtime, compiler, and OS combination is supported?
- **Operational boundary:** Which scheduling, transport, monitoring, and disconnected-installation paths are qualified?
- **Acceptance responsibility:** Who supplies reproducible test evidence and resolves each open blocker?

Require procurement claims to point to the packet's workload and tests. GPU Smith's first-party description frames the engineering recommendation as its product; an adjacent advisor's role in this analysis is to organize evidence and acceptance decisions, not certify undocumented compatibility. (gpusmith.com)

Maintain a dual-build and reopening policy

Keep a validated CUDA build available while the HIP path is qualified. Route both through continuous integration (CI), with shared correctness requirements and backend-specific build and deployment checks. Reopen findings when source, dependencies, compiler flags, target architecture, or runtime packages change. This proposed policy follows the importance of environment capture rather than assuming an earlier translation diff remains sufficient. (reproducible-builds.org)

Port incrementally and test each converted section. Preserve warnings even when the application appears to run. Keep “currently blocked” scoped to the selected requirements and date; a later package or implementation may change the disposition, but it does not retroactively validate today's procurement assumptions.

For air-gapped installations, rehearse package acquisition and installation in a disconnected test environment before acceptance. Capture everything needed to rebuild and operate the selected configuration. Docker's digest identifies an image independently of its tag, but the image record must accompany the host-driver and device-access record. (docs.docker.com)

Conclusion

CUDA to ROCm porting should begin with an inventory whose units, omissions, and closure requirements are explicit. A useful assessment separates source translation from library replacement, compiler changes, numerical behavior, performance tuning, and operational packaging. The resulting answer is a set of reviewable findings for a specified application and target stack.

HIPIFY is valuable because it can produce candidate transformations and diagnostic evidence. Its usefulness should be measured against the source and compilation settings actually examined. A clean-looking output file cannot resolve an unavailable package, an embedded instruction assumption, or an untested communication path. Those findings need their own acceptance evidence.

The proposed classifier helps distinguish mechanical work from manual portability, substitutions, redesign, and present blockers. It supports a migration-complexity count while preserving criticality, ownership, and missing evidence. It does not convert row counts into labor estimates or imply that all findings have equal business impact.

The final go/no-go decision should rest on reproducible builds, declared correctness tolerances, representative performance, and a qualified deployment configuration. Keep a validated fallback until those conditions are met, and preserve the evidence needed to repeat the decision after software changes. For engineering leaders and infrastructure buyers, that packet makes the migration scope concrete enough to evaluate alongside hardware, networking, and operating requirements.

Source links

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

1. https://rocm.docs.amd.com/projects/HIPIFY/en/latest/
2. https://github.com/ROCm/HIPIFY
3. https://rocm.docs.amd.com/en/latest/about/release-notes.html
4. https://docs.jax.dev/en/latest/installation.html
5. https://onnxruntime.ai/docs/execution-providers/ROCm-ExecutionProvider.html
6. https://cmake.org/cmake/help/latest/variable/CMAKE_EXPORT_COMPILE_COMMANDS.html
7. https://docs.nvidia.com/cuda/inline-ptx-assembly/

8. https://docs.pytorch.org/docs/2.14/cpp_extension.html
9. https://mlcommons.org/benchmarks/endpoints/
10. https://gpusmith.com/
11. https://gpusmith.com/articles/en/ai-accelerator-comparison-nvidia-amd-intel-cerebras
12. https://gpusmith.com/articles/en/rocm-10-production-upgrade-runbook
13. https://reproducible-builds.org/docs/definition/
14. https://rocm.docs.amd.com/projects/HIPIFY/en/latest/reference/supported_apis.html
15. https://rocm.docs.amd.com/projects/HIPIFY/en/latest/how-to/hipify-clang.html
16. https://rocm.docs.amd.com/projects/HIPIFY/en/latest/how-to/hipify-perl.html
17. https://rocm.docs.amd.com/projects/HIP/en/latest/how-to/hip_porting_guide.html
18. https://rocm.docs.amd.com/projects/HIPIFY/en/latest/reference/hipify-clang-cmd.html
19. https://cmake.org/cmake/help/v3.30/prop_tgt/HIP_ARCHITECTURES.html
20. https://cmake.org/cmake/help/v3.30/variable/CMAKE_HIP_ARCHITECTURES.html
21. https://github.com/ROCm/pytorch/wiki/Enabling-PyTorch-CUDA-extensions-for-ROCm
22. https://llvm.org/docs/AMDGPUUsage.html
23. https://github.com/ROCm/rocm-libraries/tree/develop/projects/hipblas
24. https://github.com/ROCm/rocm-libraries/tree/develop/projects/hipfft
25. https://github.com/ROCm/rocm-libraries/blob/develop/projects/hipsparse/README.md
26. https://github.com/ROCm/rocm-libraries/tree/develop/projects/hiprand
27. https://github.com/ROCm/rocm-libraries/tree/develop/projects/hipcub
28. https://github.com/ROCm/rocm-libraries/tree/develop/projects/miopen
29. https://github.com/ROCm/rccl
30. https://rocm.docs.amd.com/projects/MIOpen/en/latest/MIOpen_Porting_Guide.html
31. https://docs.pytorch.org/docs/2.14/notes/hip.html
32. https://www.tensorflow.org/install/source
33. https://docs.cupy.dev/en/stable/install.html
34. https://triton-lang.org/main/programming-guide/chapter-3/debugging.html
35. https://github.com/triton-lang/triton
36. https://docs.mlcommons.org/inference/submission/
37. https://docs.docker.com/engine/containers/run/
38. https://onnxruntime.ai/docs/execution-providers/MIGraphX-ExecutionProvider.html
39. https://docs.open-mpi.org/en/v5.0.x/tuning-apps/networking/rocm.html
40. https://docs.kernel.org/admin-guide/cgroup-v2.html
41. https://kubernetes.io/docs/concepts/extend-kubernetes/compute-storage-net/device-plugins/
42. https://rocm.docs.amd.com/en/latest/compatibility/compatibility-matrix.html
43. https://spack.readthedocs.io/en/latest/environments.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

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