CnuasRT, Datasheet

GPU runtime library, CUDA Runtime style host API for CnuasGPU

Document DS-CNU-011 • **Revision** D • **Issued** 23 August 2026 • **Status** Published

Item	Value
Part	`libcnuasrt.so`
Type	Host runtime library
SONAME	`libcnuasrt.so.0`, current build `libcnuasrt.so.0.2.0`
Package	`cnuas-libcnuasrt`
Version	`v0.2.0-29-gf3e5f47`
Repo	PacketFive/CnuasGPU

1. Overview

CnuasRT is the host side runtime that guest applications link against to drive a CnuasGPU accelerator. It occupies the same position in the stack that the CUDA Runtime occupies for NVIDIA hardware: it owns device selection, device memory, host to device transfer, and kernel dispatch, and it hides the character device and register level detail that CnuasDev exposes.

In QEMU PCIe mode that hidden interface is `/dev/cnuasgpuN`, created by `cnuasgpu.ko` for each bound PCI function. In Soft-GPU mode CnuasDev selects an in-process arena and no device node is involved.

The library carries a fixed set of compute entry points, and alongside them a module and launch interface that runs a CnuasIR object on a backend that can execute one. Every function listed in section 4 is present in the built shared object and was read from its dynamic symbol table, not from a design document.

Key features

- CUDA Runtime style naming and call shape, so ported code reads familiarly.
- Multi device support with an explicit current device per host thread.
- Device memory allocation, release, and bidirectional copy.
- Fixed function vector and matrix kernels dispatched to a tuned backend.
- Module load and kernel launch for CnuasIR objects, with bounds checked device pointer arguments.
- Runtime backend selection across scalar, AVX2, and AVX-512 host code paths.
- String error reporting for every status code.

2. Position in the stack

The diagram below is the QEMU PCIe path. The Soft-GPU path ends at the in-process CnuasDev arena instead of continuing through a character device.

Diagram available in the online documentation at `https://github.com/PacketFive/cnuas`.

3. Functional specifications

Parameter	Value
Library version	0.2.0
ABI	C, no name mangling

Parameter	Value
Exported runtime symbols	16 public plus 3 backend introspection helpers
Device model	Multiple devices, zero based index, per process current device
Memory transfer kinds	Host to device, device to host, device to device
Kernel set	Single precision vector add, scale, dot, and general matrix multiply
Precision	32 bit floating point
Compute backends	`scalar`, `x86_avx2`, `x86_avx512`, selected at first use
Thread safety	Current device is process wide state, see section 7
Link line	`-lcnuasrt`

4. Application programming interface

Every symbol below is exported by `libcnuasrt.so.0.2.0`.

4.1 Lifecycle

Function	Purpose
`cnuasInit`	Open the driver, enumerate devices, and select the initial device
`cnuasShutdown`	Release device handles and runtime state

4.2 Device management

Function	Purpose
`cnuasGetDeviceCount`	Number of CnuasGPU devices visible to the process
`cnuasSetDevice`	Make a device current for subsequent calls
`cnuasGetDevice`	Report the current device index
`cnuasGetDeviceProperties`	Name, revision, streaming multiprocessor count, lanes per multiprocessor, tensor tile size, device memory size, firmware version

4.3 Memory

Function	Purpose
`cnuasMalloc`	Allocate device memory from the current device
`cnuasFree`	Release a device allocation
`cnuasMemcpy`	Copy host to device, device to host, or device to device
`cnuasInternalMapDevice`	Map device memory into the host address space, used by the fixed function kernels

4.4 Synchronisation

Function	Purpose
`cnuasDeviceSynchronize`	Barrier against outstanding device work

4.5 Compute

Function	Operation
`cnuasVectorAddF32`	Element wise `y = a + b` over 32 bit floats
`cnuasVectorScaleF32`	Element wise `y = alpha * x`
`cnuasVectorDotF32`	Reduction `sum(a[i] * b[i])`
`cnuasSgemm`	Single precision general matrix multiply

4.6 Modules and kernel launch

Function	Purpose
`cnuasModuleLoad`	Validate a CnuasIR object header and take a private copy of the image
`cnuasModuleUnload`	Release a loaded module
`cnuasLaunchKernel`	Run a loaded module on the current device's compute backend

An argument is either a device pointer, translated and bounds checked against the extent the caller declares, or a scalar, passed through unchanged. There is no floating point argument kind, because the CnuasIR v0.1 subset has no move between the integer and floating point register files: a kernel cannot get a float out of an argument register and has to load one from memory, so a float is passed in a device buffer.

The argument count the module header declares must equal the count the caller passes. A mismatch is refused rather than padded, since the two sides disagreeing about a signature is how a kernel writes outside the buffer it was given.

4.7 Diagnostics and backend introspection

Function	Purpose
`cnuasGetErrorString`	Human readable text for a `cnuasError` status code
`cnuas_compute_get`	Return the active compute backend vtable
`cnuas_compute_active_name`	Name of the backend the loader selected
`cnuas_compute_reset`	Drop the cached backend so the next call re-selects

5. What this runtime does not provide

Kernels compile, but only the scalar subset

`cnuasModuleLoad`, `cnuasModuleUnload` and `cnuasLaunchKernel` are implemented and run a CnuasIR object, and `cnuascc` now compiles a device source file into one, so a kernel no longer has to be written as an instruction stream by hand. What remains absent is vector code generation, a context API, and a triple angle bracket launch syntax, so a kernel that needs the vector unit is still hand written and the caller uses the module and launch entry points directly. Only the CnuasIR interpreting backend implements `execute_cnuasir`, so a launch fails with `cnuasErrorNotSupported` on any other backend. See CnuasCC and CnuasIR, and treat any launch syntax in the design documents as a target rather than a shipped feature.

Absent capability	Tracked in
Vector code generation for user written kernels	CnuasCC
Streams, events, and asynchronous copies	Roadmap
Unified or managed memory	Roadmap
Half and bfloat16 precision kernels	Roadmap, backend capability bits are already defined
Peer to peer copy across CnuasLink	CnuasLink
CnuasGPU memory registration with CnuasNIC	Cnuas Peer Memory, provided by the separate `libcnuaspeermem` API

6. Compute backend selection

`libcnuasrt.so` does not contain the arithmetic itself. Kernels live in separate shared objects that the loader picks at first use, so one runtime binary works across host generations. See CnuasDev section 5 for the backend ABI.

Environment variable	Effect
`CNUAS_COMPUTE_BACKEND`	Force a named backend instead of automatic selection
`CNUAS_BACKEND_DIR`	Directory to search for backend shared objects
`CNUAS_COMPUTE_LIST`	Print the discovered backends and exit the selection step
`CNUAS_COMPUTE_VERBOSE`	Log the selection decision

7. Operating notes

- The current device is runtime state, not thread local state. Call `cnuasSetDevice` from the thread that will issue the following calls, and do not assume independent per thread device selection.
- `cnuasDeviceSynchronize` is a barrier against work already submitted. Because the present kernels complete synchronously, it succeeds without waiting.
- Device memory is allocated by a bump pointer allocator in the kernel driver. Allocations are reclaimed when the file descriptor closes, so a long running process that repeatedly allocates and frees will still advance the pointer.

8. Validation

Feature	Test	Device required
Initialise, device count, properties, allocate, copy, free	`lib/test/cnuasrt_smoke.c`	Yes
Activation entry points, dispatch and argument rejection	`lib/test/cnuasrt_activation_smoke.c`	Yes
Matrix multiply correctness across sizes	`lib/test/sgemm_smoke.c`	Yes
Vector and matrix kernels through the runtime	`lib/test/compute_smoke.c`	Yes
Backend vtable numerics for every instruction set	`lib/test/compute_backend_smoke.c`	No
Backend shared object ABI and capability bits	`lib/test/compute_so_smoke.c`	No
Independent state across two devices	`lib/test/multi_gpu_smoke.c`	Yes, two devices

Current results are recorded in the Validation Matrix.

9. Integration information

Item	Value
Repo	`PacketFive/CnuasGPU`
Submodule path	`src/cnuasgpu`
Source	`lib/libcnuasrt/`
Header	`cnuasrt.h`
Debian package	`cnuas-libcnuasrt`
Depends on	`libcnuasdev.so`, `cnuasgpu.ko`, a CnuasGPU PCI device
Language	C
Build	Make

10. Revision history

Revision	Notes
A	First publication. Interface list taken from the dynamic symbol table of `libcnuasrt.so.0.2.0`.
B	Records roadmap 6.4c. Adds section 4.6 for `cnuasModuleLoad`, `cnuasModuleUnload` and `cnuasLaunchKernel`, and records the kernel launch path in section 5.
C	Distinguishes the separate `libcnuaspeermem` DMA-BUF RDMA API from the runtime and CnuasLink peer copy.
D	Clarifies that the illustrated PCIe path uses `/dev/cnuasgpuN`, while the separate Soft-GPU path is in-process.

PacketFive, Packet Five Networks Ltd., Dublin, Ireland.