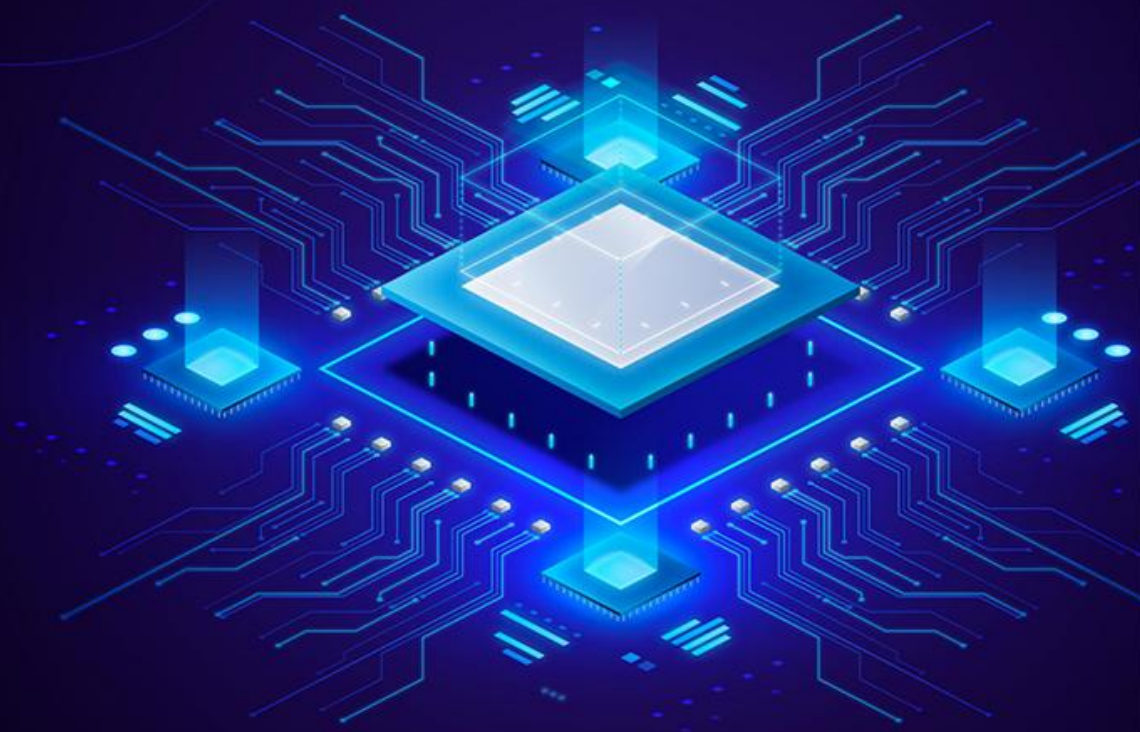
CHAKRA-GP

A Retargetable Compiler Framework for RISC-V GPGPU



Prachi Pandey
Joint Director
CDAC Bangalore

CDAC RISC-V Mission

COMPILER DEVELOPMENT FOR CDAC'S RISC-V HARDWARE

RISC-V Summit 2026

C-DAC's RISC-V HARDWARE PLATFORMS

VISHWA-AP
RISC-V HPC Processor
Vectorized & cache-efficient CPU for HPC

VISHWA-GP
RISC-V GPGPU
Massively parallel GPU for AI and HPC workloads

VISHWA-AX
AI Accelerator
Optimized for tensor/matrix workloads

DSRA
Domain Specific & Adaptive
Flexible programmable accelerators

THE COMPILER IS THE ENABLING LAYER

The compiler is as strategic as the chip itself — the critical bridge between custom silicon and real-world applications.

Enabling --

- Software frameworks to access custom hardware
- Performance portability across HPC and AI workloads
- Hardware-aware code generation and optimisation
- Rapid iteration in hardware-software co-design

OUR MISSION

Indigenous | **Unified Compiler Stack** | **Performance Optimized** | **Open & Extensible Ecosystem**

Compiler Gaps in Current RISC-V GPU Ecosystems

RISC-V Summit 2026

Popular compiler infrastructures such as **LLVM** and **GCC** provide mature support for scalar and **RVV-based** vector execution.

CURRENT LLVM/GCC CAPABILITIES

- Scalar ISA Lowering
- RVV Vectorization
- Generic LLVM Optimizations
- Conventional Memory Lowering

* Designed primarily for scalar and vector execution models

Existing RVV Support ≠ GPU Compiler Support

RVV (Vector)	SIMT (GPU)
Operates on Vector Lanes	Many threads execute in parallel(Warps)
SIMD-style Execution	SIMT Execution Model
Predicted Vector Execution	Thread-Centric with divergence
Flat Memory Model	Hierarchical GPU memory model

RVV support alone is insufficient for efficient SIMT GPU execution

REQUIRED GPU COMPILER SUPPORT

- SIMT Execution Lowering
- Warp-Level Scheduling
- Divergence Analysis & Handling
- GPU Memory Lowering
- Matrix Instruction Generation

GPU-Oriented Instruction Categories for VISHWA-GP

Specialized instruction abstractions to enable efficient GPU compilation on RISC-V

RISC-V Summit 2026

1 Divergence Instructions

Controls thread-level parallel execution and divergence handling.

- `convr`
- `divr`

2 Memory Instructions

Optimizes memory movement and hierarchical data access

- `ld.cg`
- `ld.shared`

3 Synchronization Instructions

Coordinates execution and memory ordering across threads

- `wsync`
- `memcnt`

4 Tensor Instructions

Accelerates tensor and matrix computations

- `hmma`
- `ldsm`

5 Transcendental Instructions

Special functional instructions for some mathematical operations

- `sin`
- `cos`

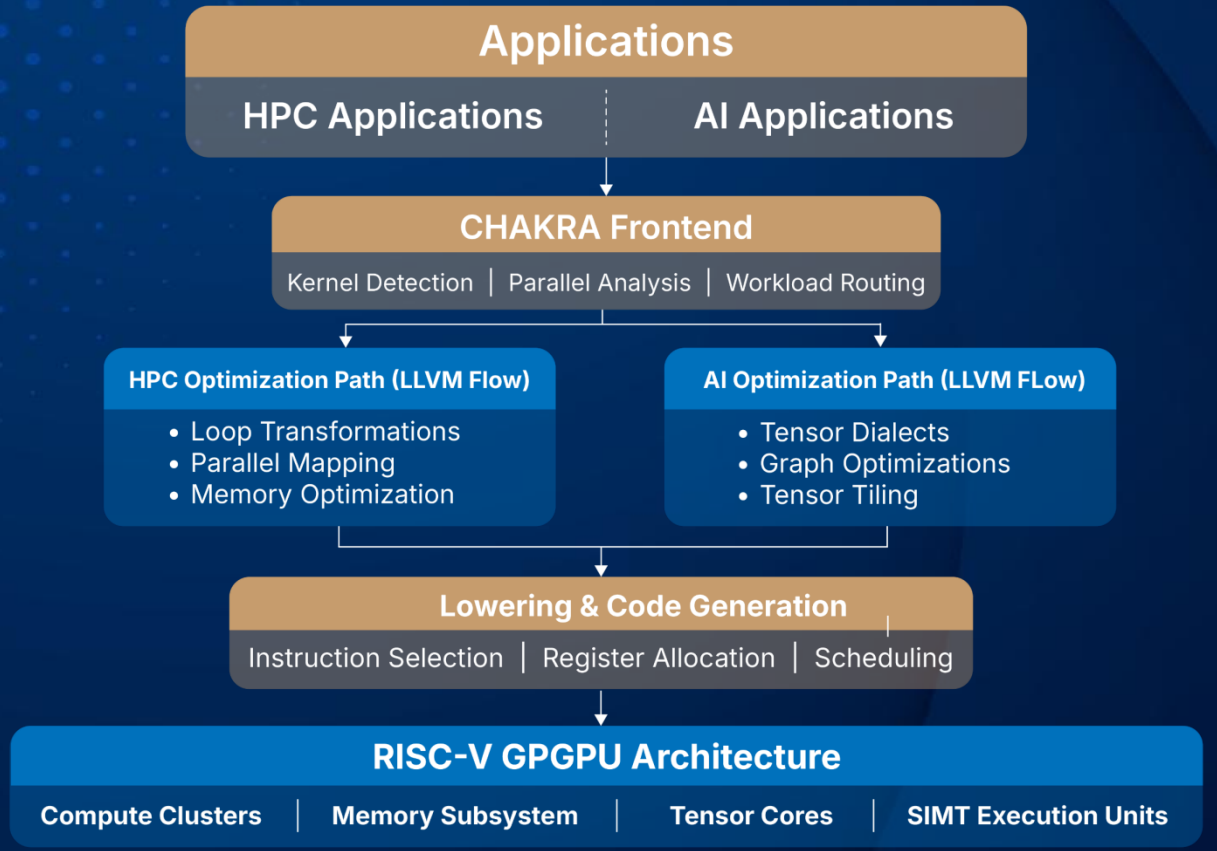
GPU-oriented instruction abstractions enable efficient SIMT execution, memory hierarchy utilization, synchronization, and tensor acceleration on RISC-V GPUs.

CHAKRA-GP: A Hybrid LLVM—MLIR Compiler Framework for VISHWA GP

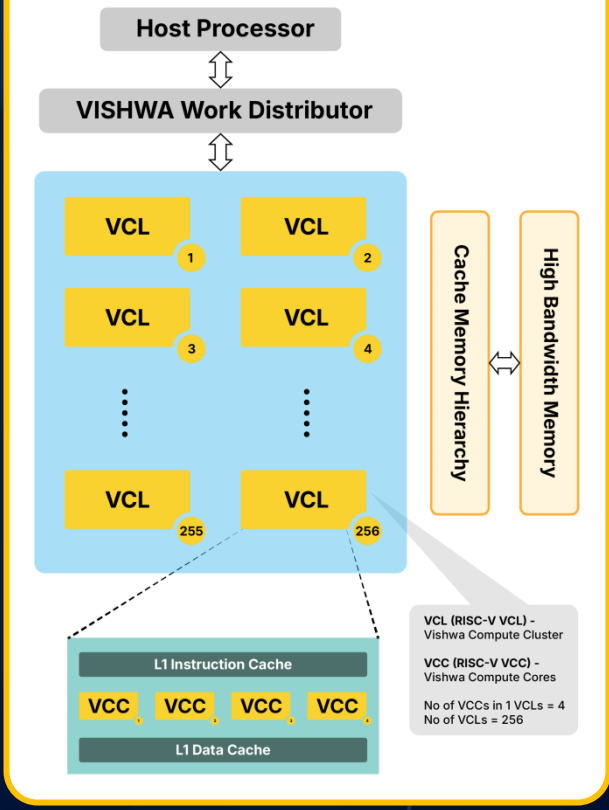
RISC-V Summit 2026

CHAKRA-GP AT A GLANCE

- Supports Multiple Frontends
- SIMT-Aware Execution
- Tensor-Aware Optimizations
- Custom Instructions
- Posit Support



TARGET: VISHWA GPU



CHAKRA-GP provides a unified, extensible, and high-performance compiler infrastructure to unlock the full potential of VISHWA and future RISC-V GPU architectures

CHAKRA-GP HPC Compilation Flow

From Device kernel to executable on VISHWA-GP

RISC-V Summit 2026

1. Host Application (CPU)

```
const float alpha)
    {
        int gid =
get_global_id(0);
        C[gid] =
alpha * A[gid] +
B[gid];
    }
```

Enqueued to Runtime

- Kernel String
- Kernel Arguments
- Work-dim/ND Range

Triggers Compilation

CHAKRA-GP

2. Host Compilation Path

gcc/clang → IR (Host Optimization) → Host Binary → Runtime Interface

3. Device Compilation Path

Clang Frontend → **LLVM IR Generation** (Generate target independent IR)

Transformed IR for VISHWA-GPGPU

Address Space Generation	. Private . Local/Shared	. Global . Constant
Thread Disambiguation	. Thread ID resolution	. Work-group mapping
Thread Synchronization	. Barriers . Atomics	. Memory Fences
Memory Semantics	. Coalescing . Cache hints	. Bank Conflict handling

CHAKRA Backend for VISHWA-GPGPU

- Thread Mapping
- SIMT aware Scheduling
- Intrinsic Lowering
- Resource Allocation
- Kernel Outlining & Launch Configuration

LLVM Dialect Lowering → **LLVM IR Generation**

Modified Clang + LLVM + Custom VISHWA GPU Backend

Kernel Binary (ELF)

4. VISHWA Runtime

- 0110 1001 1010 Receives Kernel Binary (ELF)
- Loads & JITs on VISHWA-GP
- Dispatches to Compute Units
- Executes Kernel on VISHWA-GP

The host runtime triggers CHAKRA-GP to compile device kernels into optimized RISC-V GPU binaries, which are loaded by the runtime and executed on VISHWA GP compute units

CHAKRA-GP AI Compilation & Runtime Framework

From AI Models to Tensor-core execution on VISHWA-GP

RISC-V Summit 2026

Framework Level

CHAKRA-GP Compiler (MLIR Based)

VISHWA Runtime

VISHWA GPU Hardware

Framework Runtime

Pytorch Model → torch.compile(backend="chakra")

1. FX Graph
PyTorch FX Computation Graph

2. Torch MLIR Frontend
Import to MLIR (Tensor Dialects)

3. Dialects
MLIR Tensor Representation

4. Transformations & Optimizations
Tiling
Vectorization
Bufferization

5. Mapping
Mapping to Blocks & Threads (GPU Mapping)

6. GPU Kernel Outlining
GPU Kernel Outlining

7. Convert to VISHWA Ops
Lower to VISHWA Operations

8. LLVM IR Generation
Lower to LLVM IR

VISHWA GPU ISA
VISHWA-ISA (Instruction Set Architecture)

ELF/GPU Binary

1. Load Module
Load ELF/GPU Binary & initialize module

2. Memory Management
Manage Global & Shared Memory

3. Kernel Scheduling
Work distribution & Dependency Management

4. Device Dispatch
Dispatch Kernels to VISHWA GPU

5. Execution Management
Monitor, Sync & Error Handling

SIMT Cores
Thread Execution (SIMT)

Tensor Cores
MMA Operations (Mixed Precision)

Shared Memory
On-chip Memory (Scratchpad)

L1/L2 Cache
Cache Hierarchy for Data Science

Warp Scheduler
Warp Management & Scheduling

Memory Controller
DRAM Interface & Bandwidth Management

Challenges in RISC-V GPU Compilation

GPU Compilation is fundamentally harder than scalar or vector only compilation

RISC-V Summit 2026

GPU ISA Profile Standardization

Standardized support for SIMT execution, warp operations, divergence handling, and matrix/tensor extensions is essential for portable RISC-V GPU ecosystems.

Standardization enables portability

Scalable Toolchain Ecosystem

Open GPU ecosystems require mature compiler backends, profiling tools, debugging infrastructure, and binary portability.

Toolchains enable ecosystem growth

Standard GPU Thread Model

A unified execution model for warps, thread blocks, synchronization, and cooperative execution is required for scalable GPU software development.

Instruction Width

32-bit instruction insufficient for representing GPGPU instructions containing different flags and hints

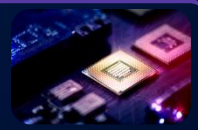
GPU-Oriented Memory Model

Shared memory semantics, synchronization behavior, atomics and address-space support require GPU-aware architectural definitions.

Mixed precision support

RISC-V lacks a standardized ISA profile for mixed-precision arithmetic (FP8, BF16, FP16, TF32, INT4/INT8) commonly required in AI/ML workloads.

Efficient GPU compilation requires coordinated optimization across execution, memory, and tensor-compute hierarchy



RISC-V Summit 2026

Thank You

prachip@cdac.in | pranoseje@cdac.in | hari@cdac.in