Nebula GPU Interconnect System: Technical Architecture and Implementation

Pranav Chandra, Pramit Pal, Meghadri Ghosh
Team Bob

September 2025

Project Overview

- **Goal:** Scalable, cache-coherent GPU interconnect for AI/ML workloads
- **Topology:** 2D mesh, 2x2 to 8x8 grid (up to 64 GPUs)
- **Protocols:** ARM AMBA AXI4 (non-coherent), CHI (coherent)
- **Implementation:** SystemVerilog RTL, Python analysis, Web dashboard
- **Key Features:**
  - Five-stage router pipeline with virtual channels
  - Adaptive and deterministic routing
  - Protocol translation bridges (AXI4/CHI $\leftrightarrow$ NoC)
  - Performance monitoring and visualization

System Architecture

- Mesh topology generated by `nebula_mesh_top.sv`
- Routers interconnected, edge nodes terminated
- Each node: router + protocol bridge + GPU/memory interface

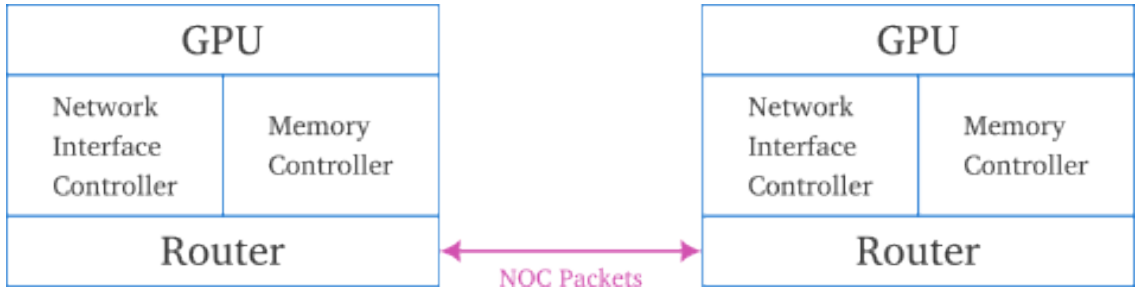

Figure: Nebula Mesh Topology Example (4x4)

Node Connections

- Nodes connect to their N, S, E & W neighbours via bidirectional links.
- The links carry flit data, control signals, and credit information.
- 2D mesh architecture, scalable from 2x2 mesh to 8x8 mesh.

Figure: Mesh Topology Example from Web Dashboard (8x8)

Router Microarchitecture

Five-Stage Pipeline:

1. Buffer Write (BW): Input port, VC allocation
2. Route Computation (RC): XY/adaptive routing
3. Virtual Channel Allocation (VA): VC state machines, credit reservation
4. Switch Allocation (SA): Crossbar arbitration, fairness
5. Switch Traversal (ST): Data transmission, credit management

Features:

- 4 VCs per port, 16-flit FIFO depth
- Credit-based flow control
- Round-robin arbitration
- Congestion-aware adaptive routing

Routing Algorithms

- **XY Routing:** Dimension-ordered, deadlock-free
- **Adaptive Routing:** Congestion-aware, selects least congested path
- **Implementation:**
  - Combinational logic compares router and destination coordinates
  - Congestion metrics: buffer utilization, credit counts
  - Programmable thresholds for adaptive switching

Technical Details

- Route Computation: lines 260-380 in `nebula_router.sv`
- VC Allocation: lines 630-690 in `nebula_router.sv`
- Switch Allocation: lines 460-570 in `nebula_router.sv`

Flit and Packet Format

- **Flit:** 256 bits, defined in `nebula_pkg.sv`
- **Header Fields:**
  - Type (HEAD/BODY/TAIL/SINGLE)
  - Source/Dest coordinates (4 bits each)
  - VC ID (2 bits)
  - Sequence number (16 bits)
  - Packet ID (8 bits)
  - QoS (4 bits)
  - CRC (32 bits)
- **Payload:** 208 bits
- **Multi-flit packets:** HEAD, BODY, TAIL; single-flit packets supported

Protocol Translation Bridges

- **AXI-NoC Bridge (`nebula_axi_noc_bridge.sv`):**
  - Burst decomposition, address mapping
  - 64-entry reorder buffer
  - Packet assembly/disassembly
  - Error detection, latency monitoring
- **CHI-NoC Bridge (`nebula_chi_noc_bridge.sv`):**
  - CHI message classification, VC mapping
  - Directory-based MOESI coherency
  - Snoop response aggregation
  - Timeout/error handling
- **Outstanding Transaction Management:**
  - Hardware table tracks up to 64 operations
  - Transaction ID, address, burst length, sequence, timeout

Packet Assembly and Disassembly

- **Assembler (`nebula_packet_assembler.sv`):**
  - Converts protocol transactions to flit packets
  - Header generation, payload segmentation
  - Address-to-coordinate mapping
- **Disassembler (`nebula_packet_disassembler.sv`):**
  - Reconstructs transactions from flits
  - CRC verification, sequence management
  - Handles out-of-order and multi-path delivery

Credit-Based Flow Control

- **Credit Controller (`nebula_credit_flow_ctrl.sv`):**
  - Per-VC credit counters, max 16
  - Increment on flit acceptance, decrement on allocation
  - Prevents buffer overflow, deadlock
  - `credits_available` signals for arbitration
- **Flow Control Protocol:**
  - Sender stalls if credits $= 0$
  - Lossless operation guaranteed

System Integration

- **Top-Level Modules:**
  - `nebula_system_top.sv`: Mesh instantiation, AXI4 interfaces, address mapping
  - `nebula_mesh_top.sv`: Router grid generation, edge handling
  - `nebula_top.sv`: Configuration registers, system status, debug
- **Address Mapping:**
  - 64-bit global address space
  - Static/dynamic schemes (partitioning, hashing)
  - Hardware decoder for coordinate extraction
- **Collective Operations:**
  - Broadcast/multicast via packet replication
  - Hardware barriers, reduction support

Verification and Testing

- **RTL Testbenches (`code/tb/`):**
  - Unit tests: packet assembler/disassembler, FIFO, arbiter
  - Integration tests: AXI-NoC bridge, router, mesh
  - System-level: end-to-end, multi-hop, congestion, performance
- **Python Tools:**
  - `nebula_traffic_generator.py`: Traffic pattern generation, testbench synthesis
  - `nebula_vcd_parser.py`: VCD trace analysis, packet event extraction
- **Metrics:**
  - Latency, throughput, congestion, error rates
  - Performance counters in routers and system top

Web Dashboard

- **Backend:** Flask + Socket.IO (`web_dashboard/backend/app.py`)
- **Frontend:** Vanilla JS + Vite + Tailwind (`web_dashboard/frontend/`)
- **Features:**
  - Real-time mesh visualization (SVG)
  - Animated packet flows, congestion heatmap
  - Performance metric graphing (utilization, latency, throughput)
  - VCD trace replay, simulation control
  - Traffic pattern selection (Uniform, Hotspot, CNN, Matrix, Transformer, etc.)
- **Integration:**
  - Runs Verilog simulations, parses VCD, updates UI via WebSocket
  - API endpoints for mesh, performance, simulation control

Performance Monitoring

- **Router-Level Counters:**
  - Packets forwarded per direction
  - Buffer utilization, VC statistics
  - Congestion, temperature (simulated)
- **System-Level Metrics:**
  - Total packets routed, average latency, throughput
  - Error and protocol violation tracking
  - Historical trending for optimization
- **Visualization:**
  - Time-series graphs, mesh overlays
  - VCD replay for cycle-accurate analysis

Technical Challenges

- **Scalability:** Mesh generation, address mapping for large grids
- **Coherency:** CHI protocol edge cases, directory state management
- **Adaptive Routing:** Congestion metrics, deadlock avoidance
- **Verification:** Multi-level test coverage, error injection
- **Integration:** Protocol bridge correctness, performance counter aggregation
- **Visualization:** Real-time updates, VCD parsing, UI responsiveness

Current Status and Future Work

- **Completed:**
  - Mesh topology, router pipeline, protocol bridges
  - AXI4/CHI support, packet assembler/disassembler
  - Python traffic generator, VCD parser, web dashboard
  - Verification infrastructure, performance monitoring
- **In Progress:**
  - Advanced adaptive routing, QoS features
  - Hierarchical clustering, multi-mesh support
  - Enhanced CHI edge case handling
  - Dashboard UI improvements, analytics
- **Planned:**
  - FPGA prototyping, hardware deployment
  - Deep learning workload benchmarks
  - Fault tolerance, dynamic reconfiguration

References

- SystemVerilog RTL: `code/rtl/`
- Python Tools: `code/python/`
- Testbenches: `code/tb/`
- Documentation: `docs/final_report.tex`, `docs/abstract.tex`
- Dashboard: `web_dashboard/`

Q&A

Questions?

Nebula RTL Modules: Overview

- The Nebula NoC is composed of modular, parameterized SystemVerilog RTL blocks.
- Each module is designed for scalability, protocol flexibility, and high throughput.
- This section details the design decisions, code structure, and unique features of each RTL module.

nebula_top.sv: System Integration (1/2)

Design Decisions:

- Parameterized mesh size, protocol enable/disable, and config bus widths.
- Unified configuration and status interface for software control.
- Modular instantiation of routers, bridges, and monitoring logic.

Key Code:

- Parameter and function definitions for mesh configuration.
- Synchronous logic for system control and status.

nebula_top.sv: System Integration (2/2)

Cool Features:

- System-wide error aggregation and debug trace export.
- Performance counters and status registers for real-time monitoring.
- Generate blocks for scalable instantiation.

Key Code:

- Generate loops for node instantiation.
- Status aggregation logic.

nebula_system_top.sv: System-Level Integration (1/2)

Design Decisions:

- Centralizes mesh, memory, and protocol bridge instantiation.
- Implements global address decoding and mapping.
- Coordinates system resets and status propagation.

Key Code:

- Mesh instantiation with parameterized width and height.
- Address mapping logic from global to local address space.

nebula_system_top.sv: System-Level Integration (2/2)

Cool Features:

- Flexible address mapping for heterogeneous memory systems.
- System-level reset and error handling.
- Integration hooks for simulation and testbench control.

Key Code:

- Synchronous reset and control logic.
- Error status propagation and handling.

nebula_mesh_top.sv: Mesh Topology (1/2)

Design Decisions:

- 2D grid instantiation using nested generate loops.
- Edge node handling (tie-off or wrap-around).
- Parameterized for mesh width/height.

Key Code:

- Generate loops for X and Y dimensions of the mesh.
- Conditional logic for edge node connections.

nebula_mesh_top.sv: Mesh Topology (2/2)

Cool Features:

- Automated port wiring for all mesh neighbors.
- Edge detection logic for boundary nodes.
- Supports mesh reconfiguration for fault tolerance.

Key Code:

- Port connection logic using generate and if-else constructs.
- Tie-off or wrap-around logic for edge cases.

nebula_router.sv: Router Pipeline (1/2)

Design Decisions:

- Five-stage pipeline for high throughput.
- Virtual channel allocation and credit-based flow control.
- Adaptive and deterministic routing support.

Key Code:

- Synchronous logic for each stage of the router pipeline.
- Combinational logic for route computation and switch allocation.

nebula_router.sv: Router Pipeline (2/2)

Cool Features:

- Per-port and per-VC performance counters.
- Error detection for buffer overflow and protocol violations.
- Round-robin arbitration for switch allocation.

Key Code:

- Performance counter increment and error flag logic.
- Arbitration logic for switch allocation.

nebula_axi_if.sv: AXI4 Interface (1/2)

Design Decisions:

- Parameterized AXI widths for flexible integration.
- FIFO-based buffering for all AXI channels.
- Clock domain crossing support.

Key Code:

- AXI signal assignments and FIFO push/pop logic.
- Clock domain crossing synchronizers.

nebula_axi_if.sv: AXI4 Interface (2/2)

Cool Features:

- AXI burst handling and response reordering.
- Backpressure propagation to the NoC.
- Error injection for verification.

Key Code:

- Logic for burst-to-packet conversion and response assembly.
- Error detection and handling logic.

nebula_axi_noc_bridge.sv: AXI4-NoC Bridge (1/2)

Design Decisions:

- Decomposes AXI bursts into NoC packets.
- Maintains reorder buffer for out-of-order responses.
- Tracks outstanding transactions by ID.

Key Code:

- Logic for AXI to NoC packet conversion.
- Reorder buffer management for response assembly.

nebula_axi_noc_bridge.sv: AXI4-NoC Bridge (2/2)

Cool Features:

- Out-of-order response assembly from NoC flits.
- Error detection and reporting for protocol violations.
- Latency monitoring for each transaction.

Key Code:

- Logic for assembling responses from NoC flits.
- Error and latency monitoring logic.

nebula_chi_interface.sv: CHI Protocol (1/2)

Design Decisions:

- Implements CHI request, response, and data channels.
- State machine for protocol sequencing.
- Snoop and directory support.

Key Code:

- CHI protocol state machine implementation.
- Logic for handling CHI requests and responses.

nebula_chi_interface.sv: CHI Protocol (2/2)

Cool Features:

- FIFO-based buffering for all CHI channels.
- Protocol error detection and recovery.
- Integration with directory and snoop logic.

Key Code:

- Buffer management logic for CHI channels.
- Error handling and recovery logic.

nebula_chi_noc_bridge.sv: CHI-NoC Bridge (1/2)

Design Decisions:

- Maps CHI message types to NoC VCs.
- Handles coherency, snoop, and directory state transitions.
- Aggregates snoop responses and manages timeouts.

Key Code:

- Logic for mapping CHI messages to NoC virtual channels.
- Snoop response aggregation and timeout handling logic.

nebula_chi_noc_bridge.sv: CHI-NoC Bridge (2/2)

Cool Features:

- Timeout detection for snoop responses.
- Priority mapping for CHI traffic classes.
- Error handling for protocol mismatches.

Key Code:

- Timeout detection and handling logic.
- Priority encoding logic for CHI messages.

nebula_chi_directory.sv: Directory Controller (1/2)

Design Decisions:

- Implements MOESI directory for cache coherency.
- Handles snoop requests and state transitions.
- Manages race conditions and protocol corner cases.

Key Code:

- Directory state machine implementation for MOESI.
- Logic for handling snoop requests and responses.

nebula_chi_directory.sv: Directory Controller (2/2)

Cool Features:

- MOESI state machine for each cache line.
- Snoop response aggregation and forwarding.
- Error detection for invalid state transitions.

Key Code:

- State transition logic for MOESI protocol.
- Logic for aggregating and forwarding snoop responses.

nebula_niu_axi.sv: AXI NIU (1/2)

Design Decisions:

- Buffers and arbitrates between protocol and NoC domains.
- Handles packetization and depacketization.
- Manages local error and status reporting.

Key Code:

- Logic for buffering and arbitration between AXI and NoC.
- Packetization and depacketization logic.

nebula_niu_axi.sv: AXI NIU (2/2)

Cool Features:

- Local error detection and reporting.
- Arbitration between multiple protocol sources.
- Parameterized for different AXI widths.

nebula_packet_assembler.sv: Packet Assembly (1/2)

Design Decisions:

- Converts protocol transactions to NoC flit packets.
- Segments payloads across multiple flits.
- Generates flit headers with CRC and sequence info.

Key Code:

- Header construction logic for different packet types.
- Payload segmentation and CRC generation logic.

nebula_packet_assembler.sv: Packet Assembly (2/2)

Cool Features:

- Address-to-coordinate mapping for mesh routing.
- CRC generation for each packet.
- Handles both single-flit and multi-flit packets.

nebula_packet_disassembler.sv: Packet Disassembly (1/2)

Design Decisions:

- Reconstructs protocol transactions from flits.
- Verifies CRC and sequence numbers.
- Handles out-of-order and multi-path delivery.

Key Code:

- Logic for reassembling transactions from received flits.
- CRC verification and error handling logic.

nebula_packet_disassembler.sv: Packet Disassembly (2/2)

Cool Features:

- Out-of-order packet reassembly.
- Error flagging for CRC mismatches.
- Sequence number tracking for reliability.

nebula_pkg.sv: Global Package (1/2)

Design Decisions:

- Centralizes all parameter, type, and enum definitions.
- Ensures consistency across all modules.
- Provides utility functions for coordinate mapping, CRC, etc.

Key Code:

- Parameter and type definitions for NoC and protocol.
- Utility function implementations (e.g., CRC calculation).

nebula_pkg.sv: Global Package (2/2)

Cool Features:

- Parameterized mesh and flit widths.
- Strong type safety for protocol and NoC structures.
- Utility functions for address/coordinate conversion.

nebula_top_simple.sv: Minimal Top (1/2)

Design Decisions:

- Minimal mesh and protocol integration for bring-up.
- Used for unit and smoke tests.

Key Code:

- Minimal mesh instantiation and AXI interface.
- Basic configuration and status logic.

nebula_top_simple.sv: Minimal Top (2/2)

Cool Features:

- Fast simulation for basic connectivity checks.
- Easy to extend for new protocol bridges.

common/: Reusable Modules (1/2)

Design Decisions:

- Implements core NoC utilities: CRC, FIFO, credit flow, arbitration.
- Parameterized for width, depth, and protocol.

Key Code:

- CRC calculation function and FIFO module.
- Credit flow control and round-robin arbiter.

common/: Reusable Modules (2/2)

Cool Features:

- Round-robin arbitration for fairness.
- Parameterized FIFO for flit/packet buffering.
- Modular, reusable code for all NoC blocks.

Key Code:

- Generic FIFO read/write logic.
- Arbiter grant and request handling.