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

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?