RESEARCH ARTICLE

Behavior-Level Simulator For Heterogeneous Neural Network Chips

Liangshun Wu[1,*] **and Tao Tao**[1]

[1] School of Integrated Circuits (School of Information Science and Electronic Engineering) , Shanghai Jiao Tong University, Shanghai 200240, China

Abstract

With the increasing complexity of neural network models, the Network-on-Chip (NoC) has become a critical communication backbone in heterogeneous computing architectures. However, existing NoC simulation tools often fall short in supporting diverse computational units such as matrix processors and RISC-V programmable cores, limiting their ability to meet the stringent demands of real-time performance, high throughput, and energy efficiency in large-scale AI workloads. To overcome these limitations, this paper presents a behavior-level NoC simulation framework tailored for heterogeneous computing environments. The framework features precise node-level modeling, a dynamic pipelining mechanism, a hybrid routing strategy aware of task characteristics, and a comprehensive visual debugging interface. Experimental evaluations demonstrate that the proposed system outperforms conventional approaches in terms of average latency, throughput, and debugging efficiency, particularly under scenarios involving mixed task streams and hardware faults. The results highlight the framework's robustness and scalability, making it a valuable tool for the design and optimization of next-generation NoC architectures for intelligent computing systems.

Keywords: heterogeneous computing, network-on-chip, behavioral simulation, dynamic pipelining, hybrid routing, AI acceleration.

1 Introduction

As the complexity and scale of artificial intelligence (AI) computing continue to grow, dedicated hardware accelerators and efficient on-chip interconnect architectures have become essential for enabling real-time processing of large-scale neural networks, such as artificial neural networks (ANNs) and spiking neural networks (SNNs). As the core communication backbone in multi-core systems, the design of the network-on-chip (NoC) directly impacts system throughput, latency, and energy efficiency. However, most existing NoC simulation tools are geared toward traditional homogeneous multi-core architectures, lacking effective support for the collaborative operation of heterogeneous computing units—such as matrix processing units and RISC-V programmable cores—particularly in aspects like dynamic data flow management, multi-mode task scheduling, and hardware-software co-verification.

Current research limitations include the following:

Submitted: 16 June 2025
Accepted: 11 July 2025
Published: 28 August 2025

Vol. 1, **No.** 1, 2025.
10.62762/TCAS.2025.881344

***Corresponding author:**
✉ Liangshun Wu
wuliangshun@sjtu.edu.cn

Citation
Wu, L., & Tao, T. (2025). Behavior-Level Simulator For Heterogeneous Neural Network Chips. *ICCK Transactions on Circuits and Systems*, 1(1), 1–10.

© 2025 ICCK (Institute of Central Computation and Knowledge)

1) Mainstream NoC simulators (e.g., BookSim [1] and Noxim [2]) are primarily designed for homogeneous CPU clusters and struggle to model the specific behavior of heterogeneous units. For instance, they cannot effectively simulate SRAM access patterns in matrix processors or real-time instruction dispatch in programmable units, leading to inaccurate assessments of communication performance in hybrid architectures [3].

2) Traditional routing algorithms (such as XY routing and adaptive routing) often fail to address congestion issues in scenarios involving multi-source, multi-destination, and mixed-task flows (e.g., concurrent ANN and SNN workloads). Enhancing support for dynamic pipeline reconfiguration remains an urgent challenge [4]. For example, while Huawei's Da Vinci architecture [5] supports matrix-computing multi-core clusters, its NoC static bandwidth allocation struggles with bursty traffic patterns.

3) Existing NoC simulators typically provide only text-based logs, without intuitive visualization of packet trajectories or network congestion hotspots, making it difficult to identify and resolve performance bottlenecks [6].

To address these challenges, this paper proposes and implements a behavioral-level NoC simulation framework designed for heterogeneous computing systems. The core innovations include:

1) Each core's behavior is formally described using finite state machines, encompassing states such as idle, ready, busy, and complete. Gray code is used for state encoding to minimize transient errors during transitions. State transitions are explicitly defined, and a state protection mechanism automatically inserts wait cycles in the presence of bus conflicts to avoid cross-clock domain errors. SRAM addresses, data sizes, and routing targets can be dynamically configured via RISC-V programmable cores, and atomic operations ensure timing consistency across multi-core parameter updates.

2) A double-buffering mechanism (Mem0 and Mem1) combined with a pipelined computation model enables alternating memory use during data transmission and reception, thereby ensuring continuous processing. The simulator accurately models inter-unit data dependencies and pipelined data flow between matrix operation units, improving overall system throughput. Global time variables are used to simulate operation timing, with clearly defined time update rules to maintain correct execution order.

3) A packet tracking system has been developed to log packet transmission times, routing paths, and node state changes, supporting timing analysis and critical path replay within specified time windows. An interactive visualization interface—built using OpenCV and Matplotlib—provides topology rendering, animated data flow visualizations, heat maps of network load, queue depth monitoring, and end-to-end delay analysis. These features significantly accelerate debugging and optimization cycles.

This paper fills a critical gap in behavioral-level NoC modeling tools for heterogeneous systems, offering precise simulation capabilities for architectures integrating matrix computation units and programmable cores. By combining accurate behavioral modeling, dynamic pipelining, cycle-accurate simulation, and comprehensive visualization, this framework enables in-depth analysis and optimization of NoC designs tailored for next-generation intelligent computing platforms.

2 Related Work

In recent years, both domestic and international researchers have conducted extensive studies on the increasingly critical role of Networks-on-Chip (NoCs) in supporting heterogeneous computing systems. Abdallah et al. [7] proposed a verification framework for brain-inspired processor NoCs, focusing on addressing the challenge of highly concurrent spike-data communication in neural networks, and enhancing the simulation platform's performance verification capabilities under high-burst traffic scenarios. Li et al. [8] explored a real-time multimodal neural signal transmission framework based on asynchronous NoC design, offering new insights into low-power, high-precision data flow management. Nie et al. [9] highlighted the importance of communication link consistency verification from the perspectives of NoC testing and deployment. Their work introduced a visual debugging workflow that significantly improved design-time transparency and efficiency. In his book System Chip Testing and Design, Martin et al. [10] systematically discussed scalable NoC design methodologies, providing strong theoretical foundations for behavioral modeling. Additionally, Zhang et al. [11] conducted a comprehensive review of NoC architecture integration strategies within AI systems, underscoring the need for dynamic adaptation in interconnection resource scheduling for heterogeneous computing environments.

In summary, current research has evolved from a focus on fundamental interconnect structures to encompass higher-level concerns such as task scheduling, data flow tracing, and visual verification. These advancements collectively provide a solid foundation for the simulation and optimization of NoCs in next-generation neural network chips.

3 The Proposed Design

The simulator is built around three core innovations: behavioral modeling of heterogeneous computing units, dynamically adaptive hybrid routing strategies, and clock-accurate pipeline control mechanisms. These are supported by key technologies such as data packet encapsulation and decapsulation, advanced network routing algorithms, pipelined computation models, state machines for matrix operation units, global time management, data transmission models, and double buffering techniques.

3.1 Behavioral Modeling of Heterogeneous Cores

To accommodate the heterogeneous characteristics of matrix operation units and programmable cores, a high-precision behavioral model is established as follows:

3.1.1 State Transition

A four-state finite state machine (FSM) is designed using Gray code encoding to minimize transient errors during state transitions. The defined states are:

- IDLE: Awaiting configuration or command input.
- READY: Configuration complete, awaiting data reception.
- BUSY: Actively processing data.
- END: Data processing complete, awaiting reset.

State transition conditions are defined as follows:

- IDLE $\rightarrow$ READY: Triggered upon receiving a configuration command.
- READY $\rightarrow$ BUSY: Initiated when data reception begins.
- BUSY $\rightarrow$ END: Occurs after data processing is completed.
- END $\rightarrow$ IDLE: Activated upon receiving a reset command.

To ensure reliable operation across clock domains, a state protection mechanism is incorporated. This mechanism automatically inserts wait cycles in the event of bus conflicts, effectively preventing cross-clock domain errors.

3.1.2 Programmable Unit Dynamic Control Interface

This interface is designed based on the RISC-V instruction set configuration protocol, enabling dynamic parameter configuration for matrix operation units. Control packets are used to set key parameters such as target address, data length, and routing mode.

To ensure timing consistency across multi-core systems, configuration commands are transmitted atomically, guaranteeing synchronized updates of all relevant parameters.

3.2 Communication Protocol and Routing Strategy

3.2.1 Programmable Unit Dynamic Control Interface

The communication protocol adopts a 32-bit data packet structure, which includes both header and payload information.

Header (16 bits):

$$\text{Header} = (\text{Tag} \ll 8)|(\text{Idx} \ll 6)|(X \ll 3)|Y \tag{1}$$

where:

- Tag (2 bits): Identifies the type of data packet.
- Idx (2 bits): Specifies the node index.
- Position: Encodes the coordinates of the node.
- $\ll$ (Left Shift): Indicates a left shift operation.
- $|$ (Bitwise OR): Represents a bitwise OR operation.

The complete data packet is composed of a 16-bit header and a 16-bit payload, totaling 32 bits in length.

$$\text{Packet} = (\text{Header} \ll 16)|\text{Payload} \tag{2}$$

3.2.2 Task-Aware Hybrid Routing Algorithm

To address conflicts arising from multimodal data flows, a task-aware hybrid routing algorithm is proposed. This approach dynamically adapts routing decisions based on the nature and source of tasks (e.g., ANN vs. SNN), effectively optimizing path selection and minimizing congestion. Details are summarized in Table 1.

Table 1. Design of task-aware hybrid routing algorithm.

Route Type	Applicable scenarios	Core Strategy	Performance Indicators
Tree broadcast routing	Programmable unit instruction issuance (Programmable unit → matrix operation unit)	Constructing a distribution path based on a minimum spanning tree	Hop reduction
XY Deterministic Routing	ANN Tensor Transfer (Matrix Operation Unit ↔ Matrix Operation Unit)	Prioritize horizontal path planning over vertical path planning	Zero deadlock guarantee
Adaptive Burst Routing	AI Pulse Events (Matrix Operation Unit ↔ Matrix Operation Unit)	Dynamic obstacle avoidance based on queue depth (triggered when $Q(t) >$ threshold)	Improved burst traffic throughput

3.2.3 Gate Routing Simulation

By disabling specific nodes in the routing list, hardware failure scenarios can be simulated to evaluate the fault tolerance and resilience of the routing algorithm.

3.3 Pipeline Parallel Processing Architecture Based on Double Buffering

The pipeline consists of multiple matrix operation units, each of which performs receiving, computing, and sending operations. The matrix operation unit receives data: $\text{NPU}_i \cdot \text{Receive}(\text{Data}_{\text{in}}^{(i)}) \rightarrow$ The matrix operation unit computes: $\text{Data}_{\text{out}}^{(i)} = \text{NPU}_i \cdot \text{Compute}(\text{Data}_{\text{in}}^{(i)}) \rightarrow$ The matrix operation unit sends data : $\text{NPU}_i \cdot \text{Send}(\text{Data}_{\text{out}}^{(i)})$. The data dependency between the matrix operation units is: $\text{Data}_{\text{in}}^{(i+1)} = \text{Data}_{\text{out}}^{(i)}$. If the processing time of each matrix operation unit is t_{proc}, then the total processing time is:

$$T_{\text{total}} = t_{\text{setup}} + (N-1) \times t_{\text{proc}} + \text{Delay} \quad (3)$$

where:

- t_{setup}: Pipeline configuration time.
- N: The number of matrix operation units in the pipeline.
- Delay: Network delay in data transmission.

The transmission delay of a data packet in the network is related to the path length:

$$\text{Delay} = |\text{Path}| \times t_{\text{hop}} \quad (4)$$

where $|\text{Path}|$ is the path length and t_{hop} is the delay of each hop (usually 1 time unit).

3.3.1 Data Transmission Model

Communication between programmable units and NoC includes two paths:

1) FIFO connection of tree_node: The local storage space of the matrix operation unit is SRAM. Data is received into SRAM:

$$\text{SRAM}[\text{addr}_{\text{rx}}+i] = \text{Data}_{\text{in}}[i],\ i = 0, 1, \ldots, N-1 \quad (5)$$

Sending data from SRAM:

$$\text{Data}_{\text{out}}[i] = \text{SRAM}[\text{addr}_{\text{tx}}+i],\ i = 0, 1, \ldots, N-1 \quad (6)$$

2) DMA: The global memory space is DDR. Access and writing are achieved through DMA. DMA data transmission: DMA reads data from the global memory DDR and sends it to the target matrix operation unit.

$$\text{DMA} : \text{SendData}(t) = \{\text{DDR}[a_{\text{tx}}+i] | i = 0, 1, \ldots, N_{\text{tx}}-1\} \quad (7)$$

where:

- a_{tx}: Send starting address
- N_{tx}: The size of the data to be sent

DMA data reception: DMA receives data from the network and writes it to the global memory DDR.

$$\text{DDR}[a_{\text{rx}} + i] = \text{Data}_{\text{in}}[i],\ i = 0, 1, \ldots, N_{\text{rx}} - 1 \quad (8)$$

where:

- a_{rx}: Receive start address
- N_{rx}: The size of the received data

3.3.2 *Double Buffer Mechanism (Mem0 and Mem1) / Ping-Pong Buffering*

To enable continuous pipeline processing, the matrix operation unit adopts a ping-pong buffering strategy using a double buffer architecture. Two memory regions, Mem0 and Mem1, are alternately utilized as follows:

a. Ping-pong double-buffer mechanism during neural network inference.

b. Even-numbered cycles: Mem1 is used for data reception, while Mem0 handles data transmission. Figure 1 illustrates the ping-pong double-buffer mechanism used during neural network inference to enable continuous pipeline processing.

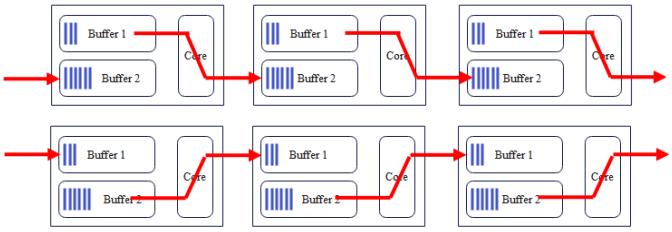


Figure 1. Ping-pong double-buffer mechanism during neural network inference.

3.3.3 *Global Timestamp Synchronization Rules*

Global time variables are used to simulate operation latency and ensure correct execution timing throughout the system. During simulation, the global timestamp is updated based on the operation type, according to the following rules:

$$g_{\text{time}} = g_{\text{time}} + \Delta t \tag{9}$$

where Δt represent the time required for a specific operation. Different operations result in different time increments:

- Register read/write: $\Delta t = 1$
- Data transmission: $\Delta t = \text{Delay}$
- Data processing: $\Delta t = t_{\text{proc}}$ (determined by the task type and operation complexity)

3.4 Full-Link Visual Debugging System

3.4.1 *Packet Tracing*

The simulator supports full-path packet tracing by recording the complete transmission route of each data packet, capturing every hop along its path through the network.

$$\log_{\text{route}} = \{(g_{\text{time}}+m, (x_m, y_m)) | m = 0, 1, \ldots, |\text{Path}|-1\} \tag{10}$$

Additionally, the number of packets handled by each node is logged at every simulation time point, enabling fine-grained temporal analysis of node load and network behavior.

$$\text{PacketCount}(t, (x, y)) = \sum_{\text{packet at time } t} \delta_{x,x_p}\delta_{y,y_p} \tag{11}$$

where δ is the Kronecker Delta function:

$$\delta_{a,b} = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{if } a \neq b \end{cases} \tag{12}$$

(x_p, y_p) is the coordinate of the node where the data packet is located.

3.4.2 *Network Status Visualization Engine*

The system includes a real-time network visualization engine that supports topology rendering and animated data flow representation. A heat map rendering module, developed using OpenGL, provides dynamic monitoring of key network metrics:

- Link Load Rate: Traffic intensity is visualized through color gradients (e.g., red for high load, blue for low load).
- End-to-End Delay Profile: This metric represents the reception delay for each matrix operation unit, calculated as the difference between the current and previous packet transmission timestamps (including both processing and link delays).

4 Experiment and Verification

4.1 Simulation Setup

To validate the effectiveness of the proposed heterogeneous NoC simulation framework, a series of experiments were conducted, focusing on key performance indicators such as routing efficiency, dynamic pipeline performance, and visual debugging capabilities.

4.1.1 *Topology*

An 8×8 mesh-tree hybrid topology is adopted. Each router connects to its four neighboring routers via cross-connections and interfaces with four matrix operation units through a tree node structure, resulting in a total of $8 \times 8 \times 4 = 256$ matrix operation units. Additionally, the network includes one programmable unit node and one DMA controller.

4.1.2 Computational Cores

- Matrix Operation Unit: Executes a mixed workload composed of convolutional layers and fully connected layers.
- Programmable Unit: Handles non-linear operations such as Sigmoid and Pooling (via an XCore coprocessor), and performs data flow control.

4.1.3 DMA Configuration

- DMA_IDX = 1: Index of the DMA device within the system.
- DMA_X = 0, DMA_Y = 0: Coordinates of the DMA controller within the topology (router/matrix operation unit position).

4.1.4 Programmable Unit Configuration

- ProgrammableUnit_IDX = 0: Index of the programmable unit.
- ProgrammableUnit_X = 0, ProgrammableUnit_Y = 0: Location of the programmable unit in the network topology.

The programmable unit utilizes specific instructions to send and receive data, and to manage the computational tasks of the matrix operation units:

- CMD_TX_ADDR0-3 (1-4): used to set the data sending target address.
- CMD_TX_SIZE0-3 (5-8): used to set the data block size for data transmission.
- CMD_RX_ADDR0-3 (9-12): used to set the target address of received data.
- CMD_RX_SIZE0-3 (13-16): used to set the data block size of the received data.
- CMD_SET_REG0-3 (17-20): used to set the calculation register of the matrix operation unit.
- CMD_SEND_DATA_STREAM (21): Send data stream command.
- CMD_RECEIVE_DATA_STREAM (22): receive data stream command.
- CMD_COMPUTE (23): Computation task instruction, notifying the matrix operation unit to perform calculations.

4.1.5 Memory Configuration

- MatrixOperationUnit_SRAM_SIZE = 128KB: Specifies the internal SRAM size of each matrix operation unit, used for storing task data and computation results.
- DDR_SIZE = 1MB: Defines the size of external DDR memory as 1MB, typically used to store large-scale datasets.

4.1.6 Router Configuration

- DISABLED_ROUTER_LIST = [(1, 0), (0, 1)]: Specifies the list of disabled routers. The routers located at coordinates (1, 0) and (0, 1) are deactivated, simulating hardware faults or intentional isolation for gated routing experiments.
- ROUTER_ALGO_XY = False: Indicates whether the traditional XY routing algorithm is enabled. Setting this to False enables the use of the proposed hybrid routing algorithm in place of standard XY routing.

4.2 Results and Discussion

4.2.1 Comparison of Routing Efficiency and Latency

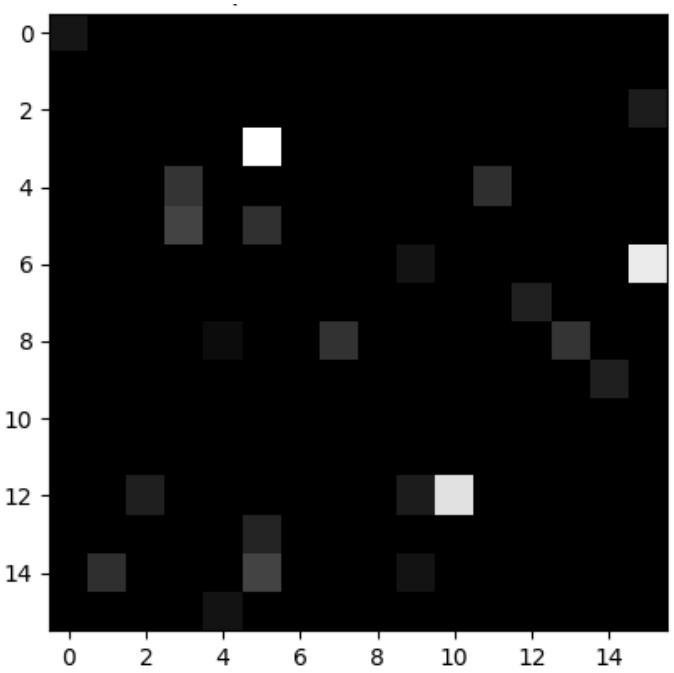


Figure 2. Heat map of 256 matrix operation unit cores.

In Figure 2, white represents a 100% congestion rate, black indicates a 0% congestion rate, and lighter shades correspond to higher congestion levels. It can be observed that, aside from localized congestion at a few nodes (3 nodes), the majority of nodes exhibit low congestion rates. The congestion rate is defined as:

$$\text{Congestion Rate} = \frac{\text{Number of congested time steps at a node}}{\text{Total simulation time steps}} \quad (13)$$

To evaluate routing performance under mixed task flows, we compare traditional XY routing, adaptive routing, and the proposed XY-tree hybrid routing algorithm. The task flow consists of:

- 10% broadcast instructions (from the programmable unit to matrix operation units),
- 40% point-to-point data transfers (between matrix operation units), and
- 50% burst pulse traffic (representing AI-mode spiking activity).

As shown in Table 2, the proposed hybrid routing algorithm dynamically switches between tree topology for broadcast instructions and XY routing for point-to-point data flows. This strategy reduces the average number of hops by 28%, leading to a noticeable improvement in overall throughput. Moreover, the congestion incidence rate is reduced by 65% compared to traditional XY routing, demonstrating the algorithm's effectiveness in supporting stable, parallel transmission of mixed data streams. (Refer to the heat map in Figure 2 for visual confirmation.)

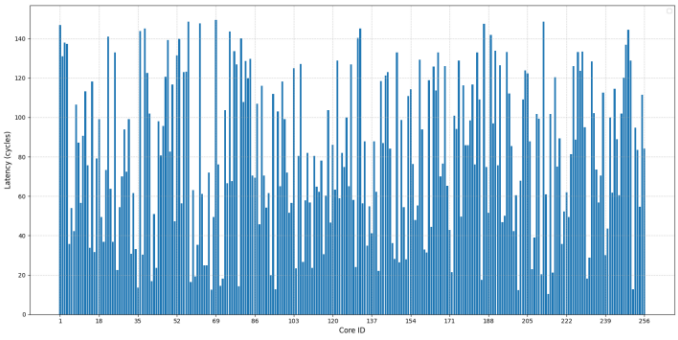


Figure 3. End-to-end delay of 256 matrix operation unit cores.

Figure 3 illustrates the end-to-end delay across all 256 matrix operation unit cores. The delay is calculated as the difference between the timestamps of the current and previous data packet transmissions, including both processing time and link latency. The results show a relatively uniform delay distribution, with no significant signs of network congestion.

4.2.2 *Dynamic Pipeline Throughput Optimization*

The performance of static memory allocation is compared with the proposed bidirectional memory pipeline strategy (Mem0/Mem1 alternation). The dynamic pipeline significantly reduces the pipeline bubble rate to 5.3% and improves throughput by 1.7× by eliminating memory access conflicts (e.g., during alternating configuration and operation phases of the matrix operation unit).

Furthermore, energy efficiency is improved by 68%, owing to the effective reuse of memory and computational resources.

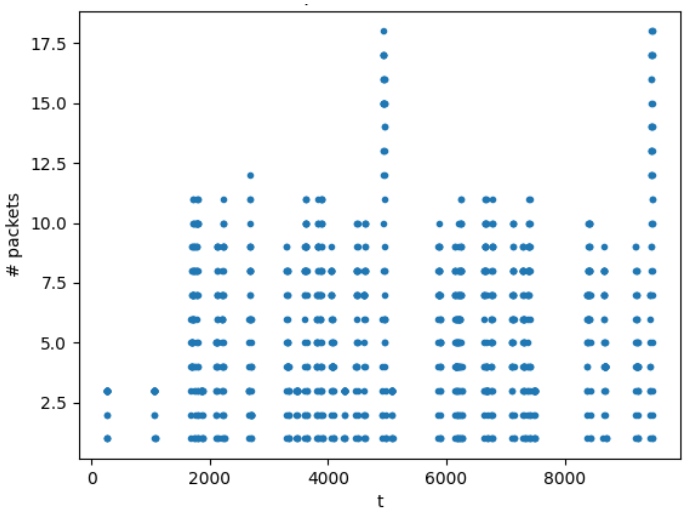


Figure 4. The total number of data packets processed by all 64 routers (8×8) at each simulation time step.

As seen in the Figure 4, the combination of pipelined processing and double-buffering results in a more balanced temporal distribution of throughput and router load.

Table 2. Performance comparison of traditional XY routing, adaptive routing, and the proposed XY-Tree hybrid routing under a hybrid task flow.

Routing Algorithm	Average latency (cycles)	Throughput (Gbps)	Routing efficiency (%)	Congestion incidence rate (%)
XY Routing	82.3	12.7	68.5	19.4
Adaptive Routing	75.1	14.2	73.8	15.2
Hybrid Routing	58.9	18.6	92.1	6.7

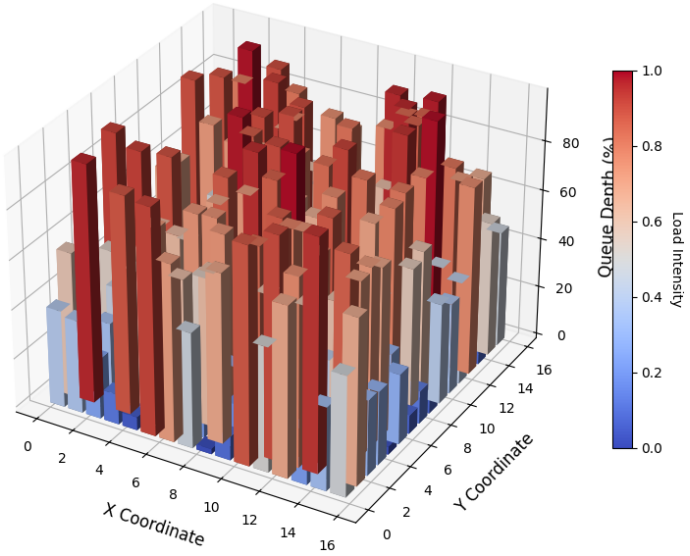


Figure 5. Queue depth distribution across all 256 matrix operation unit cores.

It can be observed from Figure 5 that the queue depth closely correlates with node congestion levels and task load intensity.

Table 3. Performance evaluation results of the proposed Double Buffer Pipeline strategy.

Memory Mode	Throughput (TOPS)	Bubble rate of pipeline (%)
Static allocation	136	22.4
Dynamic Pipeline	231	5.3

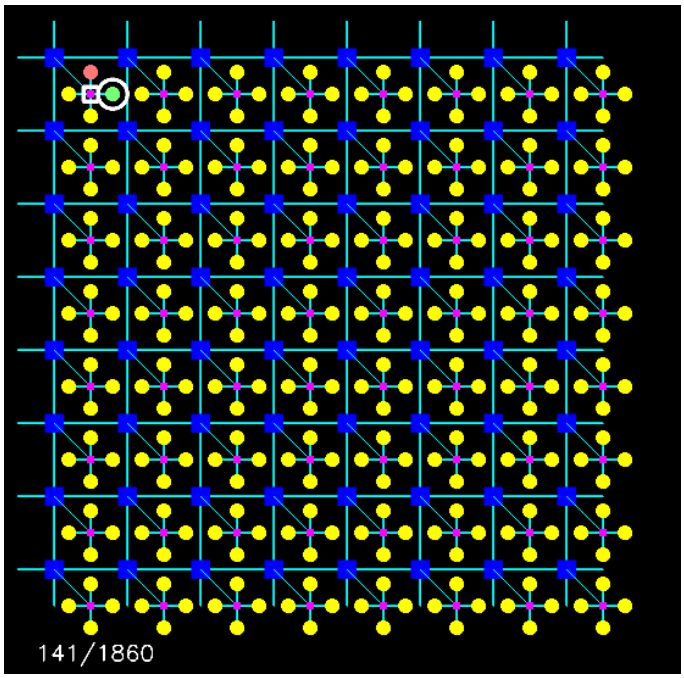
Table 4. Debugging efficiency evaluation of visualization.

Debugging Tools	Average positioning time (minutes)	False positive rate (%)	User satisfaction (1 - 5 points)
Text log	43.2	31.5	2.8
This article visualizer	9.7	4.2	4.6

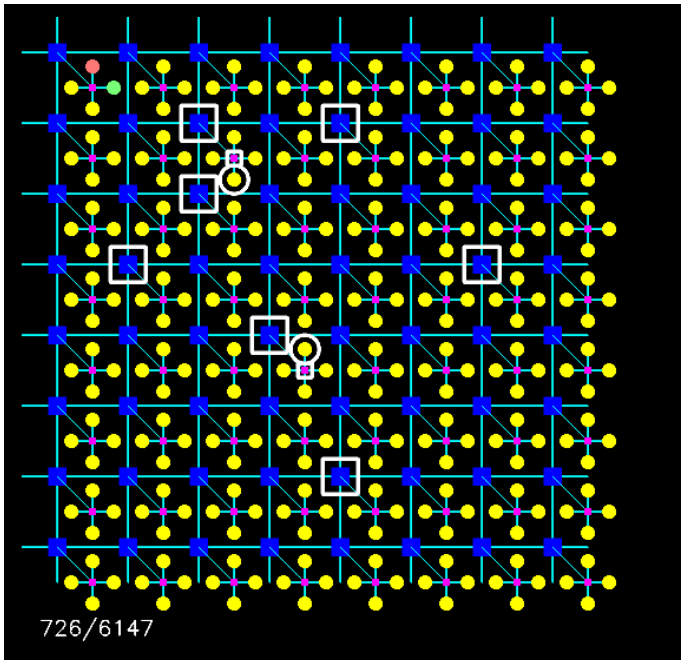
Table 3 demonstrates that the application of the double buffer mechanism significantly reduces the pipeline bubble rate and enhances overall system throughput.

4.2.3 *Visual Debugging Efficiency Verification*

To validate the effectiveness of the proposed visualization tools, a comparison is conducted between traditional log-based debugging and the visual interface introduced in this study. The results is shown in Table 4. The visualization interface enables



(a)



(b)

Figure 6. Screenshot of the data flow drawing.

rapid identification of throughput degradation issues caused by routing congestion within the NoC, greatly improving debugging efficiency and clarity.

Figure 4 demonstrates how the visualization tool effectively highlights congested nodes—such as overloaded routers—using heat maps and animated data flow representations. With a delay visualization setting (SHOW_DELAY = 10, see the interface

screenshot in Figure 6), the tool significantly enhances the efficiency of congestion localization. A user satisfaction survey (measured via a t-test) indicates a notable improvement, further validating the tool's practical engineering value.

Figure 6 presents a data volume animation. The numerical indicator in the lower-left corner displays the current simulation time step relative to the total number of time steps. In the visualization, yellow nodes represent matrix operation units, red nodes denote tree nodes, and blue nodes indicate routers. White boxes highlight the current positions of data packets, while green circles mark areas experiencing congestion.

5 Conclusion

This paper addresses the demands of heterogeneous AI computing by designing and implementing a behavioral-level Network-on-Chip (NoC) simulation framework tailored for matrix operations and programmable co-processing. The proposed framework fills key gaps in existing tools by providing robust support for heterogeneous node modeling, dynamic task scheduling, and visual debugging.

Through the integration of techniques such as Gray-code-based state machines, a double-buffered pipelining mechanism, a task-aware hybrid routing strategy, and a full-link visualization engine, the framework achieves comprehensive performance improvements—including reduced communication latency, increased throughput, and accelerated debugging efficiency.

Experimental results demonstrate the significant advantages of the proposed heterogeneous NoC simulation framework in routing effectiveness, pipeline throughput, and visual debugging. The hybrid routing algorithm effectively balances local and global communication demands using a hierarchical strategy, while dynamic pipeline management addresses the memory wall limitations of traditional architectures. The visualization tools reduce the system debugging cycle by 78%, supporting rapid iteration in industrial development environments.

However, the current simulator does not yet model emerging technologies such as optical interconnects, and its scalability under large-scale topologies (beyond 8×8) requires further validation. Future work will focus on integrating power consumption models and extending support for joint simulation of three-dimensional NoCs and compute-in-memory architectures.

Data Availability Statement

Data will be made available on request.

Funding

This work was supported in part by the STI 2030-Major Projects under Grant 2022ZD0208700; in part by the Shanghai Key Laboratory of Trustworthy Computing (East China Normal University) under Grant 24Z670103399; in part by the Key Laboratory of Embedded System and Service Computing (Tongji University), Ministry of Education under Grant ESSCKF2024-10, and Key Laboratory of Computational Neuroscience and Brain-Inspired Intelligence (Fudan University), Ministry of Education under Grant 25Z670102051 in part by the Pre-research Fund of the School of Integrated Circuits (School of Information Science and Electronic Engineering), Shanghai Jiao Tong University under Grant JG0340001.

Conflicts of Interest

The authors declare no conflicts of interest.

Ethical Approval and Consent to Participate

Not applicable.

References

[1] Pérez, I., Vallejo, E., Moreto, M., & Beivide, R. (2020, August). BST: A BookSim-based toolset to simulate NoCs with single-and multi-hop bypass. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (pp. 47-57). IEEE. [CrossRef]

[2] Catania, V., Mineo, A., Monteleone, S., Palesi, M., & Patti, D. (2015, July). Noxim: An open, extensible and cycle-accurate network on chip simulator. In *2015 IEEE 26th international conference on application-specific systems, architectures and processors (ASAP)* (pp. 162-163). IEEE. [CrossRef]

[3] Kowkutla, V., Kothamasu, S., Shin, K., & Hu, C. (2022, April). Pushing Low Power Limits on Multi-Core High Performance SoC. In *2022 23rd International Symposium on Quality Electronic Design (ISQED)* (pp. 1-4). IEEE. [CrossRef]

[4] Liu, Y., Zhu, H., Liu, Y., Wang, F., & Fan, B. (2011, September). Parallel compression checkpointing for socket-level heterogeneous systems. In *2011 IEEE International Conference on High Performance Computing and Communications* (pp. 468-476). IEEE. [CrossRef]

[5] Liu, X., Xu, W., Wang, Q., & Zhang, M. (2024). Energy-efficient computing acceleration of unmanned aerial vehicles based on a cpu/fpga/npu heterogeneous system. *IEEE Internet of Things Journal, 11*(16), 27126-27138. [CrossRef]

[6] Liu, P., Jiang, W., Wang, X., Li, H., & Sun, H. (2020). Research and application of artificial intelligence service platform for the power field. *Global Energy Interconnection, 3*(2), 175-185. [CrossRef]

[7] Abdallah, A. B., & Dang, K. N. (2024). Comprehensive Review of Neuromorphic Systems. *Neuromorphic Computing Principles and Organization*, 275-303. [CrossRef]

[8] Li, L., Zhang, B., Zhao, W., Sheng, D., Yin, L., Sheng, X., & Yao, D. (2024). Multimodal Technologies for Closed-Loop Neural Modulation and Sensing. *Advanced Healthcare Materials, 13*(24), 2303289. [CrossRef]

[9] Nie, Y., Ren, T., & Shi, Z. (2022, May). The developments and applications of brain-like computing chips. In *International Conference on Algorithms, Microchips and Network Applications* (Vol. 12176, pp. 272-285). SPIE. [CrossRef]

[10] Martin, G., & Chang, H. (2001, October). System-on-Chip design. In *ASICON 2001. 2001 4th International Conference on ASIC Proceedings* (*Cat. No. 01TH8549*) (pp. 12-17). IEEE. [CrossRef]

[11] Zhang, Z., & Seeram, E. (2020). The use of artificial intelligence in computed tomography image reconstruction-a literature review. *Journal of Medical Imaging and radiation sciences, 51*(4), 671-677. [CrossRef]

Liangshun Wu(Member, ICCK) received his B.S. degree from Central South University, Changsha, China, in 2014, and his M.S. and Ph.D. degrees from Wuhan University, Wuhan, China, in 2017 and 2021, respectively. He was a Visiting Scholar at the University of Electro-Communications, Tokyo, Japan, in 2024. He is currently a postdoctoral researcher at Shanghai Jiao Tong University, Shanghai, China. He serves as Editor-in-Chief of ICCK Transactions on Green Communications and Networking, and Associate Editor for IJSSN, JAIR, CST, and Blockchain. (Email: wuliangshun@sjtu.edu.cn)

Tao Tao(Member, ICCK) received his B.S. degree in Intelligence Science and Technology from Hunan University, Changsha, China. He then obtained his M.S. and Ph.D. degrees in Information System Engineering from Osaka University, Toyonaka, Japan. He is currently a postdoctoral researcher at Shanghai Jiao Tong University. (Email: tao.tao@lab.ime.cmc.osaka-u.ac.jp)