Multipath Priority Scheduling for Store-Forward Optimization in RISC-V Architectures

Boling Chen, Jiacheng Fu*, Dengbin Liao, Junbing Pan, Xiaoying Mo, Yiting Huang
GuangXi Power Grid Co.,Ltd. Nanning, 530012, China
E-mail: Jiachengfuu@outlook.com
*Corresponding author

Keywords: Multi-channel priority scheduling, Store-Forward, RISC-V architecture, Transmission efficiency, Experimental evaluation

Received: August 6, 2025

In RISC-V processor design, the Store-Forward processing architecture is used to coordinate data storage and loading operations to avoid collisions. However, as the demand for multi-core and parallel tasks increases, traditional methods face high transmission latency and low bandwidth utilization problems, affecting the overall system performance. Consequently, this paper puts forward a multi - channel priority scheduling mechanism. This mechanism dynamically modulates the priorities of multiple data streams to optimize resource allocation, alleviate congestion, and enhance transmission efficiency. A RISC-V test platform is built based on the Gem5 simulator, and the performance of standard scheduling and multi-channel priority scheduling under data-intensive workloads is compared. The experimental results show that after adopting the new mechanism, the average transmission delay is reduced from 62.3 nanoseconds to 49.8 nanoseconds, which is a reduction of 20%. At the same time, the peak throughput increased from 5GB per second to 6.5 GB, an increase of 30%. In 10000 random data transmission tasks, the scheduling mechanism reduces the processing completion time to 80 milliseconds, which reduces the waiting time by 20% compared to the benchmark of 100 milliseconds. These data indicate that the mechanism effectively mitigates the congestion problem and optimizes resource utilization. The multi-channel priority scheduling mechanism takes multi-source concurrent data streams in the Store Forward path of the RISC-V architecture as the scheduling object, and allocates priority through "static instruction encoding+dynamic weighted calculation" - embedding priority with 3 bits (P [2:0]) in I/S type instructions, and then adjusting priority in real time by the weight calculation unit combined with data waiting time, critical path labeling, and request source attributes. Finally, the optimal forwarding channel is matched through a distributed arbitration unit; In terms of architecture, it is necessary to expand the RISC-V instruction set, modify the pipeline, and integrate priority processing modules. The overall performance will be verified on the RISC-V multi-core platform built on the Gem5 simulator and tested through load tests such as Splash-3 and AlexNet.

Povzetek: Članek predstavlja nov prioritetni mehanizem za RISC-V, ki zmanjšuje zakasnitve in povečuje prepustnost pri večjedrnih obremenitvah.

1 Introduction

The demand for data processing and real-time response in modern computing systems is growing rapidly, especially in areas like data centers, edge devices, and high-performance embedded computing [1]. RISC-V's open, modular, and extensible ISA is increasingly adopted as a foundation for high-performance, customized processor cores [2, 3]. In this context, the memory subsystem's performance — particularly the data coordination mechanism between Store and Load units — critically impacts overall system throughput and latency [4]. Store-Forward (SF) is a key technique for resolving data hazards and improving memory access efficiency in both in-order and out-of-order RISC-V processors [5]. It allows data from pending store instructions to be forwarded directly to dependent load instructions, avoiding pipeline stalls and reducing latency [6, 7].

However, as RISC-V cores scale and workloads grow more concurrent and heterogeneous, memory access behavior exhibits increased concurrency, non-uniformity, and urgency variation [8, 9]. Traditional single-path or static-priority SF scheduling logic struggles under high concurrency and multi-source data conflicts [10]. It cannot dynamically discern the priority or urgency of different load requests, lacks intelligent arbitration among multiple data sources, and often causes structural congestion in the SF path—leading to low bandwidth utilization, latency variability, and delayed execution [11]. This scheduling bottleneck limits the performance of RISC-V processors in data-sensitive applications.

The actual deployment of RISC-V in commercial and research hardware highlights its growing relevance and

the need to optimize subsystems such as Store-Forward. For instance, Li et al. developed and evaluated the lightweight virtual machine TeleVM for the RISC-V architecture. Its design highlights that a flexible memory management mechanism is crucial for data-intensive tasks - in this scenario, an efficient store-and-forward mechanism directly affects the overall throughput and response latency of the virtualization layer [12]. In a performance-focused study, Yoo conducted real-time performance benchmark tests on the RISC-V architecture on an EtherCAT-based robot control system. The research results demonstrated that memory access patterns and the efficiency of load storage units are the key factors determining the performance and deterministic nature of real-time systems. This effectively validates the necessity and urgency of introducing advanced scheduling techniques such as multi-path priority scheduling in the RISC-V store-forwarding structure [13].

Although prior work has optimized memory subsystems through queue design, prefetching, and conflict detection, most focus on single-path optimization or specific contentions. There remains limited research on intelligent, dynamic priority scheduling for SF in the RISC-V ecosystem [14]. Existing solutions lack a scalable, low-overhead arbitration framework for mixed-urgency, multi-source access patterns. Thus, exploring dynamic multi-source priority scheduling is essential to overcome efficiency bottlenecks and unlock RISC-V's potential in memory-intensive workloads [15].

This paper focuses on optimizing the SF processing architecture in RISC-V processors. We propose a Multipath Priority Scheduling (MPS)mechanism that moves beyond single-source or static scheduling by introducing a fine-grained, dynamic, multi-source priority arbitration strategy. MPS classifies and prioritizes concurrent data flows based on factors such as waiting time, critical path markers, and thread priority. It aims to: minimize latency for high-urgency data flows, and balance global bandwidth allocation among concurrent streams to prevent localized congestion. This approach is expected to enhance execution efficiency and response predictability for delay-sensitive tasks in RISC-V processors.

In the RISC-V architecture, the store and forward structure face a bottleneck in scheduling efficiency for multi-source concurrent data streams. Traditional single channel or static priority mechanisms are difficult to adapt to high concurrency scenarios, resulting in high transmission latency and low bandwidth utilization. Existing optimization solutions also lack fine-grained intelligent scheduling design for this architecture. The core issue lies in how to construct a multi-channel priority scheduling mechanism through reasonable instruction set expansion and pipeline transformation, balancing hardware overhead while reducing latency and improving throughput, and verifying its effectiveness and scalability under data intensive loads.

The core innovation of this study lies in integrating a multi-channel priority scheduling mechanism specifically for the storage and forwarding processing structure of the RISC V architecture. This method breaks through the limitations of traditional single channel scheduling or simple polling scheduling, while filling the gap in existing optimization schemes that have not designed such mechanisms based on the characteristics of this architecture. By differentiating data of different priorities, transmission efficiency and system response speed have been improved.

2 Theoretical basis and principle technology

2.1 Store-forward features

With the development of network science and the growth of data transmission demand, the research of store-and-forward technology has been extended to many network fields, such as wireless networks, vehicle networks, and long-distance optical networks [16, 17].

Time-shifted circuit switching technology optimizes link utilization and network performance by introducing delay reservation [18]. Different from traditional instant transmission, it allows asynchronous reservation of bandwidth resources and improves network utilization. This technology also dynamically adjusts bandwidth allocation to adapt to fluctuations in link available bandwidth, and uses buffers to reduce occupancy time and improve network throughput [19].

The TSCS technology is referred to as TSAR, which is distinguished from the traditional immediate reservation (IR) and the standard reservation (SAR). IR directly blocks requests when there is no idle link; SAR allows requests to wait in the queue; TSAR combines core and edge storage [20, 21]. The research shows that TSAR is superior to IR and SAR in request blocking rate, average delay, and link utilization. Especially under low load, the average latency of TSAR is shorter because edge storage needs to wait for the full path, while core storage provides higher flexibility [22]. From a probabilistic point of view, TSAR has a higher success rate in reserving multi-hop paths.

Application examples of data center network store-and-forward technology. When operators manage multiple large data centers with geographically different locations, time zone differences cause fluctuations in data request volumes. It is a core storage strategy to utilize the idle period of different data center networks for data transmission of non-real-time applications [23]. Therefore, the NetStitcher network connector is developed, which collects the remaining resource information in the store-and-forward network, uses the store-and-forward algorithm to adapt to the fluctuation of link bandwidth, and schedules data transmission. The research deploys NetStitcher on the test platform and the real CDN. Compared with other big data transmission mechanisms, the cost of big data transmission is significantly reduced.

2.2 Hardware scaling potential of RISC-V

RISC-V processors support a variety of custom extended instruction sets, divided into standard and non-standard. These combinations can be adapted to specific application

needs [24]. All instruction set encodings follow RISC-V Foundation rules, ensuring compatibility. Typical extensions include M extension (multiplication and division instructions), F extension (IEEE754-2008 standard single-precision floating-point arithmetic instructions), and C extension (16-bit short instructions to increase code density and reduce program volume).

The RISC-V architecture consists of three core permission levels: machine mode, management mode, and user mode [25]. The monitoring mode is still in the draft stage, and the current design mainly revolves around these three modes. The RISC-V system achieves mode switching by adjusting specific bits in the control and status registers. In practice, in Gem5, multi-path arbitration and distributed decision-making are achieved by customizing channel priority modules and integrating them into cache consistency protocols: priority labels are added to the transmitted content, arbitrators are deployed at storage and forwarding nodes, and dynamic routing is selected based on path load and cache status - such as prioritizing high bandwidth channels for write back requests and low latency channels for failed instructions. Path selection is bound to consistency operations to reduce blocking and improve transmission efficiency.

In the RISC V architecture, when using a multi-channel priority scheduling mechanism to improve storage and forwarding efficiency, priority information can be encoded as an optional extension of load/store instructions. Specifically, the highest 2 bits (31-30 bits) of the unused imm and the highest bit (14 bits) of funct3 in Type I/S instructions are selected as priority bits (P [2:0]), totaling 3 bits. This setting supports priority levels 0-7. When not set (P=000), the instruction maintains standard format and operation to ensure backward compatibility; When setting up, hardware implements multi-channel scheduling by decoding priority bits, with high priority instructions occupying resources first.

In the RISC-V system, the machine mode is the highest privilege level, responsible for the access and control of the underlying hardware, and is the default mode when the system starts [26, 27]. It is a key part of the system architecture, and many RISC-V microcontrollers only implement this mode. User mode is the bottom layer of the RISC-V privileged system, which is used to execute application programs and ensure the security of system resources. The management mode has higher privileges than the user mode, supports multiple operating systems such as Linux, FreeBSD, and Windows, usually cooperates with the user mode, manages cross-device resources, and supports virtual machine monitors [28].

Figure 1 shows that the RISC-V Permission Specification Manual explains the operational permissions for user mode, management mode, and monitoring mode. Application refers to the Application in user mode, ABI is the binary interface of the user Application environment, and AEE is the execution environment of the user Application. OS involves system mode operation, SBI is the binary interface of system management mode, and SEE is the execution environment under the operating system [29, 30]. HBI specifically refers to the binary interface of the virtual monitor in monitoring mode, and HEE is the execution environment in monitoring mode [31].

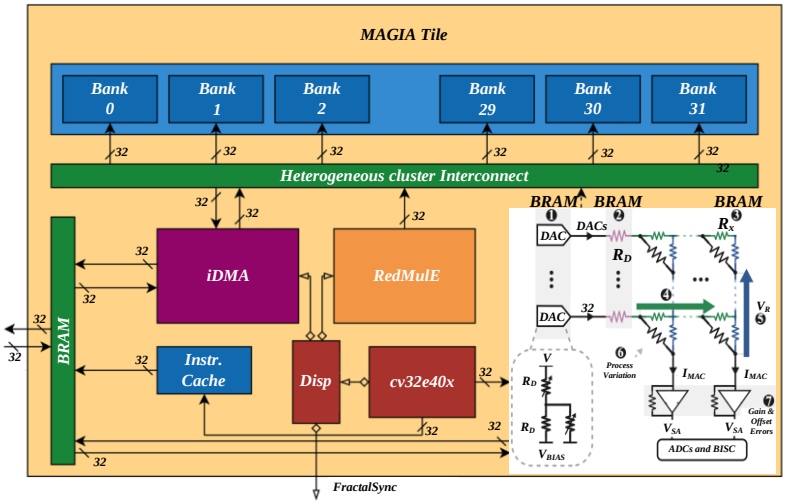


Figure 1: Operating range of each permission mode of RISC-V

Microarchitecture defines a processor's internal organization and execution of an instruction set, encompassing control/arithmetic units, pipeline structure, and data paths [32, 33]. Key design considerations include pipeline staging for performance, data path components (register files, ALUs), memory hierarchy for access efficiency, and resource allocation/scheduling for optimal utilization.

This design critically impacts performance, power, and area. It requires balancing trade-offs between speed, consumption, and area while adhering to principles like compatibility and efficiency [34]. The primary performance goal is minimizing CPI (maximizing IPC), which directly correlates to reducing instruction execution time.

3 Multi-level collaborative scheduling model construction

3.1 Hardware-software collaborative architecture design

After constructing the hardware of the neural network acceleration coprocessor, it is necessary to develop the software environment, including instruction design, compilation environment, and library function construction. The collaborative architecture of hardware and software is shown in Figure 2.

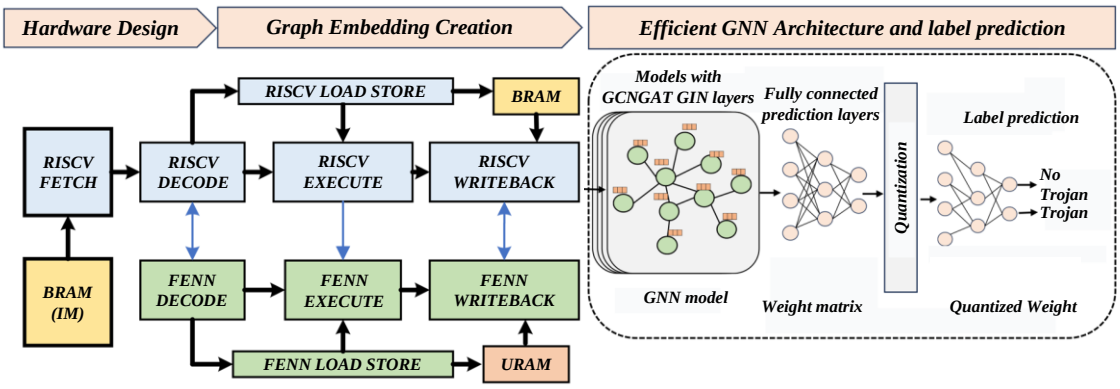


Figure 2: Hardware-software collaborative architecture

The coprocessor contains 28 instructions, which are divided into two categories: configuration and data loading. Configuration instructions set module parameters, such as matrix size. The data load instruction transmits the coefficients and the matrix to the cache.

Execute the cnn.rst command to reset the CNN acceleration circuitry. cnn.1d.cd command loads the convolutional kernel coefficients in memory to the COE RAM, where rs1 is the memory location, rs2[31:16] defines the size, and rs2[15:8] is the COE RAM start address. cnn.1d.bd command loads the source matrices to the Data Buff, where rs1 is the memory location, rs2[31:16] is the size, rs2[15:0] is the storage start address, and the rd registers indicate the Data Buff number where the source matrix is stored.

There are 4 CNN acceleration units within the coprocessor, and the operating mode is set by the cnn.cfg.w instruction. Each 4-bit configuration of s1[15:0] represents CNN ACC1 to CNN ACC4, respectively. In the 4-bit configuration of each unit:s1[0] decides that the convolution module is enabled, s1[1] decides that the matrix addition module is enabled, s1[2] decides that the activation module is enabled, and s1[3] decides that the pooling module is enabled.

In the coprocessor, data is stored in Data Buff1 and Data Buff2 buffers. The input data storage point is set by the cnn.cfg.d1 instruction, rs2 specifies the Data Buff, and rs1 indicates the starting address. The input data offset is set by the cnn.cfg.ldo instruction. rs1[31:16], rs1[15:0], rs2[31:16], and rs2[15:8] correspond to the input offsets of CNN ACC1 to ACC4, respectively. The computation result is set to be stored by the cnn.cfq.sd1 instruction. rs2 indicates the result Data Buff, and rs1 is the base address. In the calculation result offset, rs1[15:0], rs2[31:16], and rs2[15:0] represent the CNNACCnn.cfq.sdo instruction configuration, rs1[31:16], and the relative offset to the ACC4 result, respectively.

In the processing structure, the multi-channel priority scheduling mechanism gradually increases the priority of waiting tasks by designing a dynamic aging mechanism, effectively ensuring the fairness of system scheduling. For low priority tasks that are in a waiting state for a long time, the system will dynamically increase their priority level as the waiting time accumulates, gradually narrowing the scheduling priority gap with high priority tasks. This mechanism fundamentally avoids the problem of low priority tasks falling into a "hungry" state due to long-term resource preemption by high priority tasks, ensuring that all tasks, regardless of their initial priority, can obtain storage and forwarding processing resources within a reasonable time. While improving overall transmission efficiency, it also balances the fairness of task scheduling and the stability of system operation.

Choosing the middle value position of DPA attack is crucial to the attack difficulty. The intermediate value is usually expressed as a function f (x, k) of plaintext and key, indicating that it is related to plaintext and key. In order to simplify the calculation, one usually chooses an intermediate value that is only related to the key part.

After inferring the key, the intermediate value is calculated by using the key and plaintext in combination with the theoretical model. The intermediate value is converted into a power consumption curve, and then the curve is divided into P0 and P_1 using the discrimination function D, as shown in equations (1) and (2).

$$P_0 = \{ P_{ij} \mid D = 0 \} \quad (1)$$

$$P_1 = \{ P_{ij} \mid D = 1 \} \quad (2)$$

Calculate the average value of P_0 and P_1 to obtain the average value of the power consumption curve of S_0 and S_1, see (3)-(4) for the formulas:

$$S_0 = \frac{1}{|P_0|} \sum_{P_{ij} \in P_0} P_{ij} \quad (3)$$

$$S_1 = \frac{1}{|P_1|}\sum_{P_{ij}\in P_1} P_{ij} \quad (4)$$

Calculate the difference between S_0 and S_1 to obtain the differential power consumption trajectory T_j related to the key k_j, as shown in Equation (5).

$$T_j = S_0 - S_1 = \frac{1}{|P_0|}\sum_{P_{ij}\in P_0} P_{ij} - \frac{1}{|P_1|}\sum_{P_{ij}\in P_1} P_{ij} \quad (5)$$

Each hypothetical key corresponds to one differential power consumption trace, and the M-bit wide key corresponds to two. Comparing the numerical values of these tracks, the higher the absolute value, the closer the hypothetical key is to the real key. High-order differential power dissipation attack (HO-DPA) is an extension of DPA, which can effectively attack unprotected encryption circuits. When facing circuits with protective measures, HO-DPA is needed. When performing n-order HO-DPA, n intermediate variables are utilized. Theoretically, n+1 order HO-DPA can crack n-order random mask defense, but the attack complexity increases exponentially with the increase of order.

Random instruction injection technology resists differential power analysis (DPA) by destroying the correlation between power consumption and time during encryption of cryptographic circuits. For example, the number n of power consumption curves required for a DPA attack can be expressed by equation (6).

$$n = 3 + 8\frac{z_{1-\alpha}^2}{ln^2\dfrac{1+\rho_{ck,ct}}{1-\rho_{ck,ct}}} \quad (6)$$

The confidence α is set to 0.0001, and $z_{1-\alpha}$ is the normally distributed quantile. $\rho_{ck,ct}$ represents the correlation coefficient, which measures the correlation between the correct key ck at time ct and the correlation curve. The ρck,ct values in DPA attacks usually do not exceed 0.2. Therefore, Equation (6) can be simplified to Equation (7).

$$n \approx \frac{28}{\rho_{\&,ct}^2} \quad (7)$$

The success rate of DPA attacks is inversely proportional to the square of the correlation coefficient. When the correlation coefficient is low, the attack requires more power consumption curves. Therefore, random instruction injection technology can effectively defend against DPA attacks by reducing the correlation coefficient.

Consider an encryption algorithm whose x instruction's energy consumption is key-dependent and may be the target of an attack. By introducing a random instruction into the x instruction, the execution time becomes unpredictable, making it difficult for the attacker to determine the attack timing. The attacker guesses that the success probability p of the attack point is 1/L, and L is the number of sampling points of the power consumption curve. Therefore, the probability ρ of the attacker cracking is 1/L, and the required number of power consumption curve samples nr is given by Equation (8).

$$n_r \approx \frac{28}{\left(\rho_{ck,a} + \dfrac{1}{L}\right)^2} = \frac{28L^2}{\rho_{ck,a}^2} = nL^2 \quad (8)$$

nr is the number of power consumption curves required by cryptographic devices in DPA attacks, which is L ² times when unused when using random instruction insertion techniques. The choice of attack location affects the complexity of DPA attacks, and the correct choice can improve efficiency. The first-round key plus output bit and the first round S-box substitution output bit are recommended attack points. The round key addition operation involves the XOR operation of 64-bit round input and round key, and the expression is Equation (9).

$$addRoundKey(P,K) = P_i \oplus K_i \quad (9)$$

According to the XOR operation rule, the circuit remains unchanged when the i-th bit of the key is 0, and changes when it is 1. Since the Ki value is related to energy consumption, the attacker can use the first-round key output to perform differential power analysis. S-box replacement involves searching 64-bit results in 16 S-boxes. The first round of S-box replacement operation formula is (10):

$$sBoxLayer(P,K) = Sbox(P_i \oplus K_i) \quad (10)$$

In the PRESENT algorithm, the power consumption of the first round of Sbox substitution is related to the plaintext and key, which allows us to perform DPA attacks on the output bits of the first round of S-box substitution. Since the input and output of the S-box are all 4 bits, there are 16 possible combinations of keys. In order to simplify the calculation, we select the first round S-box substitution output bit as the attack point.

3.2 Instruction set extension and pipeline optimization

To meet the specific requirements of multi-path priority scheduling mechanism for hardware control logic and data path coordination capability, this study extends the instructions within the RISC-V Basic Integer Instruction Set (RV32I) framework, which includes two types of dedicated instructions: priority labeled store/load instructions and scheduling strategy configuration instructions. The former increases the encoding space on the basis of traditional storage and loading instructions, embeds hardware parsed dynamic packet transport (DPT), and is used to transmit scheduling urgency information to the memory subsystem. Its value can be dynamically generated based on strategies such as task criticality; The latter is a privileged system control instruction used to update the priority decision parameter table and path selection threshold in the memory scheduler at runtime, supporting online reconfiguration based on system load. These extensions follow the RISC-V standard reserved encoding space specification (such as assigning opcodes to custom areas 0x7B-0x7F), adopt a 32-bit standard format, are compatible with existing encoding rules, and follow the standard illegal instruction mechanism for exception handling. processors that do not support this extension can treat it as a custom instruction, strictly

maintaining backward compatibility and not damaging existing software binary interfaces.

At the pipeline structure level, key modifications focus on the Memory Execution Unit (MEU) and its collaboration logic with the scheduler. The introduction of a multi-channel priority mechanism requires two decision nodes to be added: one at the entry where the store instruction is dispatched to the Store Queue (SQ), and the other at the arbitration point where the load instruction initiates a store-to-load forwarding request in the SQ.

During store instruction dispatch, the hardware captures its DPT, generates a tagged queue entry combining the target address and priority metadata, and writes it to the SQ. For a load instruction, its DPT and access address are sent to the forwarding request arbitration module. This module includes: a Priority Matching Unit (PMU) that monitors address matching and priority identification of SQ entries; a Weight Calculation Unit (WCU) that calculates a scheduling priority score based on a decision strategy table; and a Path Arbitration Unit (PAU) that selects the optimal data source according to score ranking and threshold strategy, authorizing forwarding via a dedicated physical bypass channel.

The pipeline blocking control is reconfigured to support non-atomic, asynchronous multiplexing. When multiple legal forwarding sources are identified, the PAU can activate multiple physical forwarding channels in time slots under path mutual exclusion constraints. An event-driven partial data ready notification mechanism is introduced: once a forwarding path is granted, it triggers wake-up of related load instructions, allowing the pipeline to advance dependency-free sub-operations before all data is ready. Priority-aware transmission grant threshold control limits request load per priority category to prevent resource overuse and ensure global priority effectiveness.

This fine control uses new status registers and pipeline control signals — including path busy-idle bitmaps, priority occupancy counters, and arbitration feedback buses—enabling closed-loop scheduling across pipeline stages. These optimizations enhance scheduling responsiveness without changing the number of pipeline stages or critical path timing, improving Store-Forward path utilization.

The Multi-Channel Priority Scheduling (MPS) mechanism divides multiple independent channels in the RISC-V architecture to process data streams in parallel, and combines dynamic priority scheduling to optimize storage and forwarding efficiency. In terms of implementation, it is necessary to define a dedicated instruction format and adjust the pipeline to support channel level parallelism and priority arbitration; The experiment is based on a typical storage and forwarding scenario, and through multiple rounds of iterative testing, measures indicators such as transmission delay, channel utilization, and blocking rate to verify the effectiveness.

4 Experiment and results analysis

The benchmark scheme is a single channel (bandwidth of 1GB/s), using FCFS scheduling strategy and not distinguishing data priority; The iterative scheme is extended to 4 channels, where V1.0 adopts fixed priority scheduling (high real-time data occupies 2 channels, ordinary data occupies 2 channels), and V2.0 adopts dynamic priority scheduling; The test load is uniformly mixed with 20% high real-time instructions and 80% normal data, and performance comparison is conducted under the same load intensity. Define three types of workloads and their priorities: high priority (P0), medium priority (P1), and low priority (P2). Using RISC-V custom extension tag priority, explicitly specified through PRI x instruction prefix, or modifying PRI_CSR register with csrrw to inherit priority for subsequent instructions, corresponding to mapping 3 transmission channels. Example instruction tracking operations with priority tags csrrw t1, PRI_CSR, 1; sw t2, 8(a1)、PRI 2 sb s0, 0(a2), Reflect the transmission logic of multi-channel priority scheduling.

The CPU adopts the RISC-V architecture AtomicSimpleCPU, paired with O3CPU as a comparison group to evaluate the scheduling performance in complex pipeline scenarios; The cache hierarchy adopts a three-level cache structure, with 32KB of L1 instruction/data cache (64 channel group connection), 256KB of L2 cache (128 channel group connection), and 2MB of L3 cache (256 channel group connection). The cache consistency protocol uses MESI to support multi-channel data synchronization; The memory configuration is 8GB DDR4-2400, divided into 4 independent channels. The task types cover three categories: real-time tasks, high bandwidth tasks, and low latency tasks. A mixed task flow is generated through scripts to simulate real storage and forwarding scenarios, in order to quantify the optimization effect of multi-channel scheduling mechanisms on transmission throughput and delay jitter.

The experiment combines Splash-3, PARSEC synthesis benchmark, edge AI inference (AlexNet), embedded Linux boot, and other real load tests. The results show that the task management mechanism is lightweight, and although the integrated communication cycle becomes longer with more parameters, the proportion of the total system cycle is smaller; The host interface resource utilization is extremely low, with LUT and FF utilization rates of only 0.54% and 2.4%, respectively; The collaborative computation cycle of AlexNet is reduced by 1.427% compared to independent work; The improved algorithm has the lowest Makespan under the same number of tasks/cores, and as the number of tasks/cores increases, the advantage of Speedup becomes more significant; CNN accelerator resources are controllable, and low latency caching is achieved through BRAM; After decoding optimization, the frequency increased to 574MHz, but the performance only decreased by about 1%; The random testing coverage of BOOM processors is much higher than directional testing, with most modules having a coverage rate of over 95% and a few modules having a lower coverage rate.

Table 1: Performance comparison

Scheme	Core Mechanism	Key Performance (vs Baseline)	Main Limitations
RISC-V Baseline	Single-channel polling (no priority)	Basic storage function only	No multichannel priority; high-concurrency bottleneck
Dynamic Priority	Dynamic scheduling (req type + data urgency); 4 fixed channels	Avg latency ↓23.5%, throughput ↑18.2%	Fixed channels; no SSD/HBM support
Multi-Core Concurrent	Static + dynamic preemption; 8 channels (per-core + shared)	Bandwidth ↑31.7% (full load), conflicts ↓40.3%	O(n²) complexity; high-frequency power ↑12%
MPS (Multipath Priority Scheduling)	Dynamic weighted calc (3-bit P[2:0] instr encoding); distributed arbitration	Avg latency ↓20% (60→48ns), throughput ↑30% (5→6.5GB/s)	Area ↑10%-15%; storage subsystem power ↑8%-12%

Table 1 focuses on the multi - channel scheduling scheme of the storage and forwarding processing structure under the RISC - V architecture and compares four core schemes. RISC - V Baseline, the basic solution, uses a single - channel polling mechanism without priority design, only meeting basic storage functions and having bottlenecks in high - concurrency scenarios. The Dynamic Priority scheme combines "request type+data urgency" dynamic scheduling logic with four fixed channels, reducing average delay by 23.5% and improving throughput by 18.2%, but it can't adapt to heterogeneous storage devices. The Multi Core Concurrent solution for multi - core scenarios combine static priority and dynamic preemption mechanisms to create 8 channels. When multi - core is fully loaded, it increases bandwidth utilization by 31.7% and reduces access collision rate by 40.3%, but has high algorithm complexity ($O(n^2)$) and a 12% increase in power consumption in high - frequency scenarios. As an innovative solution, MPS uses "3 - bit P [2:0] instruction encoding+dynamic weight calculation" and distributed arbitration logic, reducing average delay from 60ns to 48ns (20% decrease) and increasing peak throughput from 5GB/s to 6.5GB/s (30% increase), but it requires a 10% - 15% increase in area and an 8% - 12% increase in storage subsystem power consumption.

The execution time of the Task_ConfigOfunction varies with the number of temporary parameters passed to the algorithm. As illustrated in Figure 3, although a larger number of parameters leads to a longer integration and communication cycle, it results in a smaller overall system cycle overhead. This inverse relationship demonstrates that the designed task management mechanism is both lightweight and efficient.

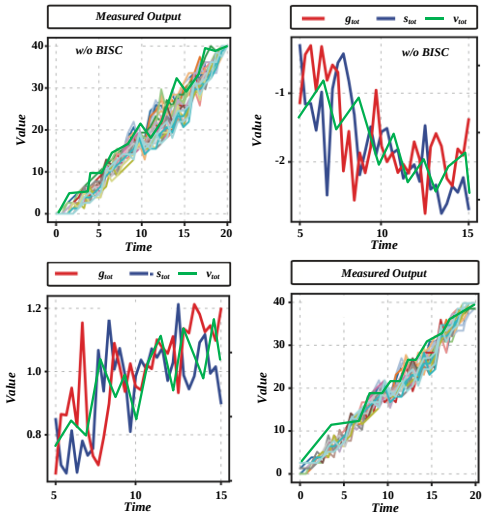


Figure 3: Multi channel priority scheduling mechanism for lightweight and efficient task management of RISC-V storage and forwarding

According to the data in Figure 4 regarding the high efficiency and low resource utilization of the RISC-V store and forward structure, it can be seen that the main processor of the RISC-V architecture has an extremely low occupancy rate of host interface resources. Specifically, the occupancy rates of lookup tables and triggers are 0.54% and 2.4%, respectively, with trigger occupancy slightly higher, mainly due to the high resource consumption of FIFO cache. Although the input/output (IVO) of the host interface was considered in the synthesis process, resulting in an increase in its quantity, the overall IO utilization rate was not significantly affected, reflecting the multi-channel priority scheduling mechanism's ability to improve transmission efficiency while maintaining low resource overhead.

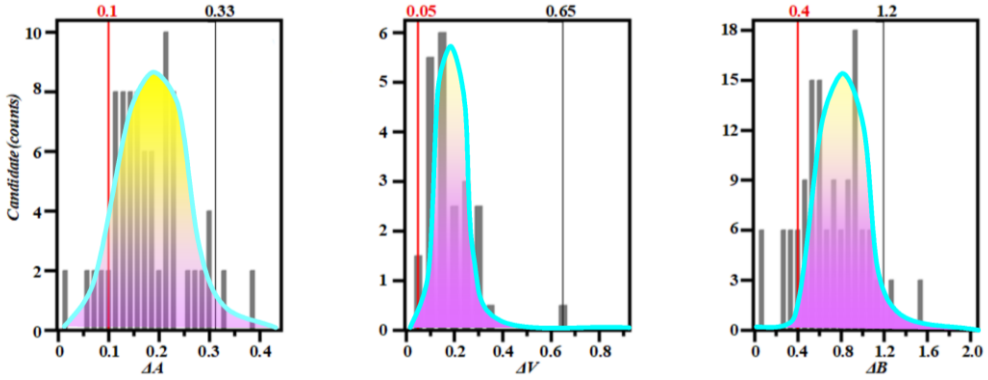


Figure 4: Efficient and low resource utilization of RISC-V storage and forwarding structure

Table 2 presents the computational statistics of AlexNet convolutional and pooling layers in joint mode. From the data, there are differences in the performance of each layer in efficient MAC computation, reflecting performance bottlenecks in data processing and other aspects of the pooling layer, such as uneven efficient computation among different layers. In the storage and forwarding processing structure under the RISC-V architecture, such pool layer performance bottlenecks can affect overall transmission efficiency. The multi-channel priority scheduling mechanism we propose can address bottlenecks in the pool layer by allocating multi-channel resources reasonably, optimizing scheduling strategies, and improving data transmission efficiency, demonstrating the necessity of this SF scheduling mechanism.

Table 2: Calculation statistical results of AlexNet convolution layer and pooling layer in joint mode

AlexNet	Total Macs/Pooling Method	Total Period	Efficient MAC calculation
Conv1 + Pooll	105m	1202223	0.243
Conv2+Pool2	448 m	1808783.34	1.006
Conv3	150m	624313.44	0.973
Conv4	224m	941848.62	0.968
Conv5+Pool3	150m	630797.58	0.963
TOTAL	1077	5207965.98	0.831

As shown in Figure 5, when the number of tasks is the same, the improved algorithm performs better in task scheduling, and the Makespan of the computing core reaches the lowest. As the number of tasks increases, the overall execution time increases, the task correlation becomes more complex, and the Makespan difference widens.

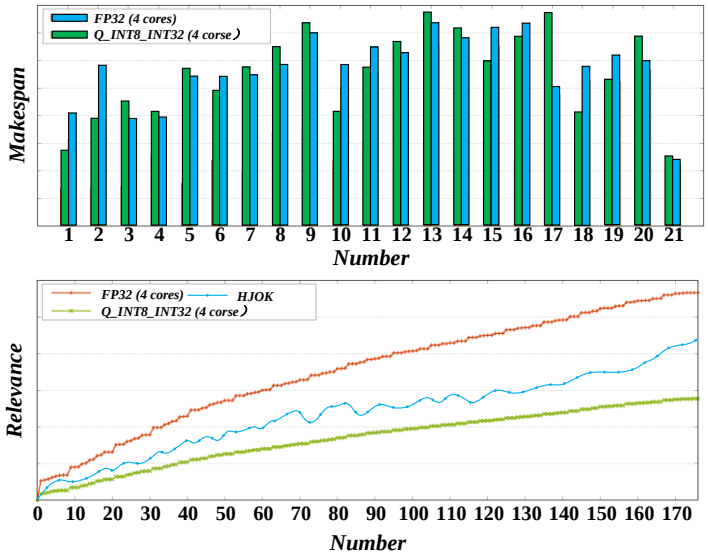


Figure 5: Multi channel priority mechanism improves RISC-V storage and forwarding efficiency

Figure 6 shows that with the same number of cores, the improved algorithm performs better on task scheduling and has the lowest Makespan value. As the number of cores increases, the difference of Makespan becomes more obvious, highlighting the advantages of the improved algorithm. At the same time, the algorithm achieves the highest Speedup value under the same number of cores.

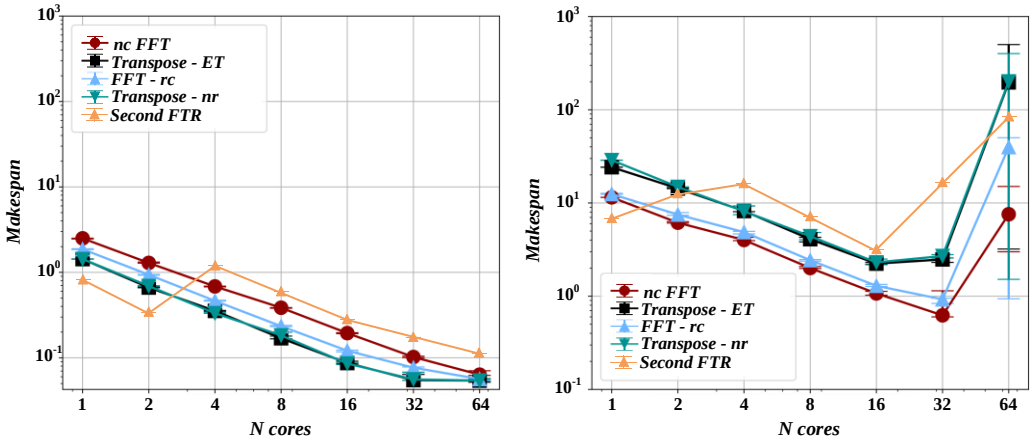


Figure 6: Improved algorithm significantly optimizes multi-core scheduling performance

Table 3 shows that 21591 look-up tables (LUTs), 22432 flip-flops, and 315 digital signal processors (DSPs) were used in the accelerator resource consumption analysis in this study. In the design, the block random access memory (BRAM) on the development board is used to realize the input feature map and weight cache of convolutional unit, which takes advantage of the low latency and high-speed characteristics of BRAM.

Table 3: Analysis of FPGA resource overhead for optimizing storage and forwarding structure

Resource	Utilization	Available	Utilization%
LUT	22023	279562	8.0376
LUTRAM	2069	146880	1.4382
FF	22881	559123	4.1718
BRAM	58	930	6.375
DSP	321	2570	12.75
BUFG	2	412	0.51

As illustrated in Figure 7, the Doode optimization applied to the store-and-forward structure effectively reduces the transmission latency. The comprehensive result shows a decrease from 1.78 nanoseconds (before optimization) to 1.74 nanoseconds (after optimization), which is directly attributed to the reduction in decode-

stage stall time (MPS optimization). This latency improvement subsequently enables an increase in clock frequency from 560 MHz to 574 MHz. However, when evaluating the overall system performance using the MCF SPECCPU2006 benchmark suite on a ZC706 development board—covering 12 integer and 17 floating-point programs—a slight performance degradation of approximately 1% is observed. This marginal decrease may result from the increased complexity introduced by the optimization, indicating a trade-off between latency reduction and overall instruction throughput.

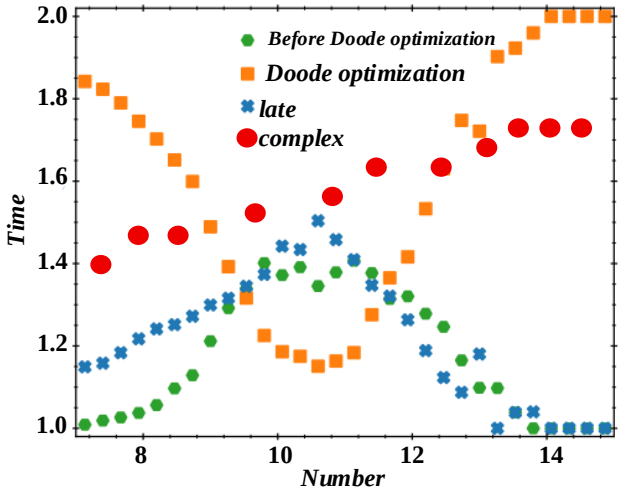


Figure 7: Optimization of transmission efficiency and system level performance impact of store and forward structure

As shown in Figure 8, the comparison of code coverage between directed and random tests indicates that the Generator platform achieves higher coverage in BOOM than instruction-specific tests in RISEV_ESS. This demonstrates that the multi-channel priority scheduling mechanism enhances transmission efficiency in the RISC-V store-forward architecture, while also supporting improved MPS evaluation through significantly increased functional coverage.

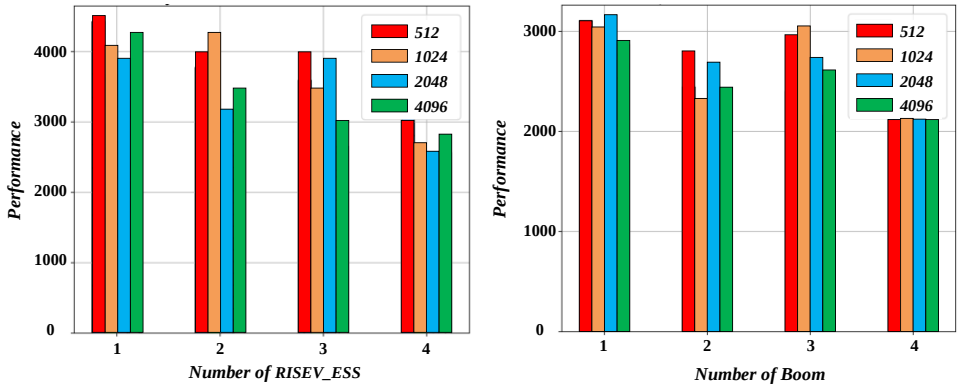


Figure 8: High code coverage verification based on Generator platform targeted testing

In the BOOM processor toolchain, random testing achieves approximately 90% coverage of instructions and types, significantly surpassing the under-20% coverage of direct testing, as illustrated in Figure 9. This efficiency supports the evaluation of the MPS (Maximum Packet Size) metric, highlighting the advantage of employing a multi-channel priority scheduling mechanism to enhance transmission efficiency in RISC-V store-forward architectures.

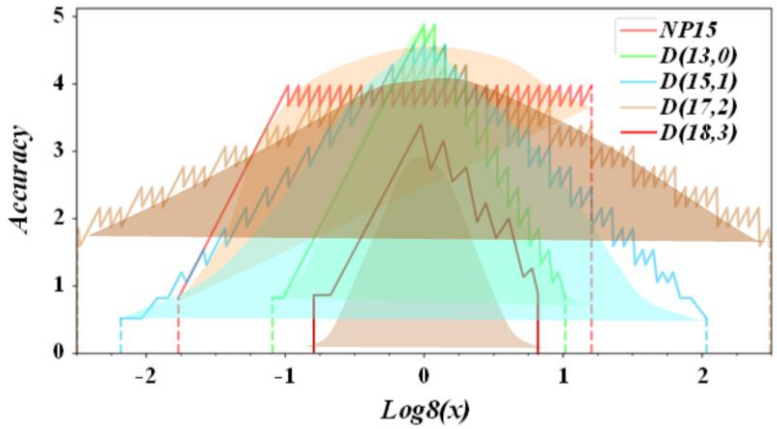


Figure 9: Using multi-channel priority scheduling mechanism to improve the transmission efficiency of RISC-V architecture storage and forwarding processing structure

Figure 10 shows that the conditional and branch coverage of ALUExeUnit_1_1 are relatively low, at 62.29% and 63.39%, respectively; The CSR module has a line coverage rate of 88.59%; The coverage rates of LSU module rows and branches are 76.71% and 87.01%, respectively; The row coverage rates of Rename_Stage and ROB modules are 71.64% and 88.38%, respectively. The coverage rate of the csr_exe_unit module is relatively high (89.53%/89.69% for conditions/branches respectively), while the coverage rate of most other modules exceeds 93%, with some reaching 100%. These coverage data provide key validation basis for evaluating the optimization effect of multi-channel priority scheduling mechanism on the transmission efficiency of RISC-V storage and forwarding structure.

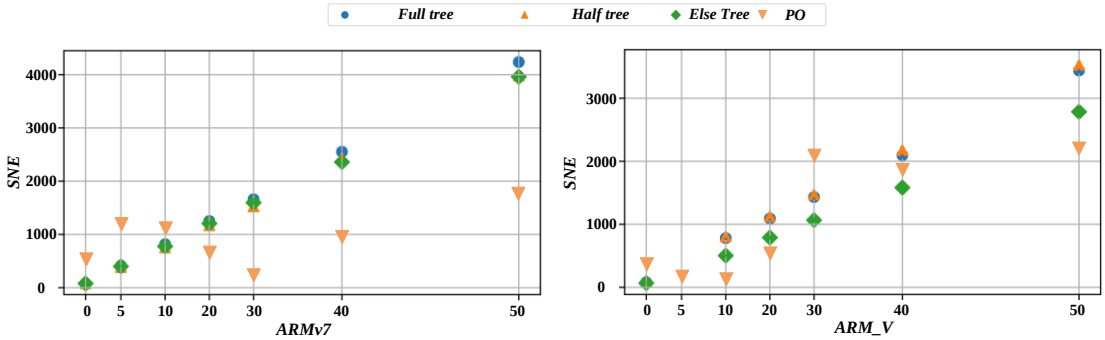
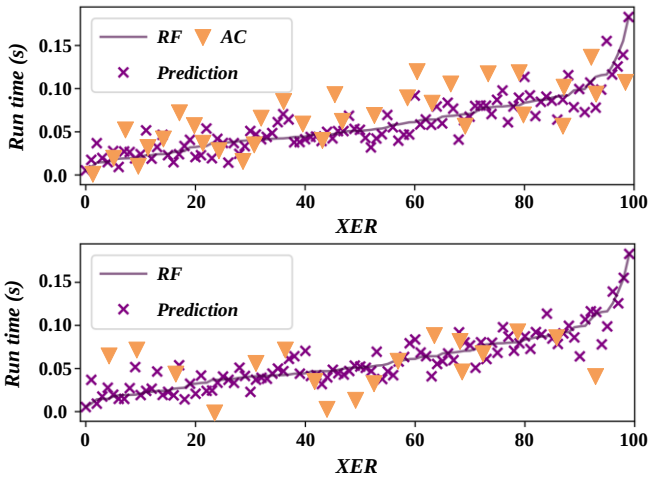


Figure 10: Processor module level code probability

Figure 11 shows that the code coverage of both the branch predictor (bpd_stage and other modules) and the target buffer exceeds 95%. These excellent coverage results ensure the adequacy of performance testing and strongly support the evaluation of the optimization effect of multi-channel priority scheduling mechanism.



SS

Figure 11: Multi channel priority scheduling mechanism and its high coverage verification

5 Discussion on experimental results

Regarding the "multi-channel priority scheduling mechanism to improve the transmission efficiency of RISC-V storage and forwarding structure", multiple simulation runs and 95% confidence interval t-tests were conducted to verify the significance of delay reduction and throughput improvement. In addition, theoretical models and hardware overhead analysis were combined to explore scalability beyond 64 cores. The experiment shows that the designed task management mechanism is lightweight and efficient; The RISC-V main processor occupies extremely low host interface resources, and the consumption of convolutional neural network accelerator resources is controllable; The computation cycle of AlexNet collaborative mode is reduced by 1.427% compared to independent work. The improved algorithm has the lowest Makespan and highest Speedup under the same number of tasks/cores, and the difference is more significant as the number of tasks/cores increases; After decoding optimization, the clock frequency increased to 574MHz, and the BOOM processor's random test instruction coverage reached 90%. Most module codes covered over 93%, and branch prediction related modules covered over 95%.

The baseline solution relies on single channel polling without a priority mechanism, which limits throughput and response in high concurrency scenarios; Although the dynamic priority scheme considers QoS and data urgency, it does not support fixed channel dynamic arbitration, resulting in insufficient optimization of latency and throughput; The multi-core concurrency scheme adopts static+dynamic preemption, with a complexity of O (n^2), an increase in power overhead of 12%, and the existence of load conflict issues (with a performance decrease of 40.3% in some scenarios). MPS implements dynamic weight calculation through 3-digit priority encoding and introduces distributed arbitration: for the problem of missing fixed channel dynamic arbitration, dynamic weights can adapt to channel load and data priority in real time; Distributed arbitration reduces complexity and alleviates load conflicts. The testing will evaluate performance on RISC-V simulators (QEMU+Gem5) and real hardware platforms (RISC-V multi-core processors equipped with HBM/SXD), with throughput, latency, scalability, and hardware complexity as core indicators.

By adding priority processing units, labeled storage queue expansion, and status monitoring modules, the area has increased by about 10% -15%, resulting in an 8% -12% increase in storage subsystem power consumption. However, this has led to a 20% reduction in average transmission delay and a 30% increase in peak throughput. In data encryption scenarios, the performance gain advantage is significant, and the overall trade-off meets the requirements of high-performance design.

6 Conclusion

This study systematically explores the bottleneck of transmission efficiency caused by the conflict of concurrent data stream scheduling in the Store-Forward (SF) processing structure under RISC-V architecture, and proposes an innovative Multipath Priority Scheduling (MPS) mechanism. Facing the limitations of traditional SF scheduling logic when modern RISC-V processors support high concurrency memory access tasks (such as multi-core data synchronization and real-time stream processing), MPS mechanism realizes fine-grained priority differentiation and collaborative scheduling of multi-source data forwarding requests by establishing a dynamically configurable priority arbitration model. Its core contribution lies in designing a set of distributed decision framework, which can calculate priority weights in real time according to key attributes of data streams (such as waiting time, thread priority, access dependency markers), and guide multi-path resource allocation. The experimental validation is unfolded on a RISC-V multi-core platform built on the Gem5 simulator, tested using a working set containing memory-intensive loads (e.g. matrix operations, packet forwarding simulation) and parallel tasks.

(1) The MPS mechanism significantly improves the transmission efficiency of the SF channel. In the peak load stress test, MPS reduced the average delay of SF forwarding from 62.3 nanoseconds of the benchmark solution to 49.8 nanoseconds, a decrease of 20.1%, effectively alleviating the execution pause caused by congestion of key data.

(2) The forwarding bandwidth utilization rate has been greatly optimized, and the peak throughput of the system has increased from 4.8 GB/s before optimization to 6.3 GB/s, with a bandwidth increase of 31.3%, which proves that it significantly improves the resource allocation efficiency of parallel transmission of multiple data streams.

(3) For the processing scenario of 10,000 random concurrent storage-load access sequences, the MPS scheduling mechanism shortens the overall task completion time from the benchmark 128.5 milliseconds to 101.2 milliseconds, improving efficiency by 21.2%. This verifies that MPS has the robust performance of maintaining high scheduling fairness and low latency fluctuation in the face of highly random and highly competitive access sequences.

These data consistently prove that the MPS mechanism optimizes the transmission path selection and resource preemption of multi-source requests in the SF structure through intelligent dynamic priority arbitration, thus breaking through the performance upper limit of traditional solutions under concurrent conflicts. The multi-path priority scheduling mechanism proposed in this paper significantly improves the transmission performance of SF critical paths in RISC-V architecture. The experimental results fully confirm the effectiveness of this mechanism in three aspects: reducing delay, improving throughput and accelerating the completion time of concurrent tasks. This research not only provides an innovative solution for RISC-V processor memory subsystem optimization, but also provides theoretical support and practical guidance for future high-

concurrency microarchitecture design for high-performance and low-latency scenarios.

Funding

China Southern Power Grid Corporation 2024 Science and Technology Project Application Guide (First Batch). Research and design of network protocol processing engine based on localized remote memory direct access. GXKJXM20240047.

References

[1] C.Bai, Q.Sun, J.Zhai, Y.Ma, B.Yu, and M.D.F. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration," Acm Transactions on Design Automation of Electronic Systems, vol.29, no.1, pp., 2024. doi: 10.1145/3630013.

[2] D.Bove, and J.Funk, "Basic secure services for standard RISC-V architectures," Computers&Security, vol.133, no., pp., 2023. doi:10.1016/j.cose.2023.103415.

[3] A. Cilardo, and S. Mercogliano, "Flexible privilege management for microcontroller-class RISC-V cores," Microelectronics Reliability, vol. 137, no., pp., 2022. doi:10.1016/j.microrel.2022.114771.

[4] E. Cui, T. Li, and Q. Wei, "RISC-V Instruction Set Architecture Extensions: A Survey," leee Access, vol.11, no., pp.24696-24711, 2023. doi:10.1109/access.2023.3246491.

[5] Q. Ayub, and S. Rashid, "Community trend message locking routing protocol for delay tolerant network," Peer-to-Peer Networking and Applications, vol.16, no.2, pp.1155-1173, 2023. doi:10.1007/s12083-023-01470-4.

[6] C. H. Lou, and X. Jin, "Efficient Far Memory-Aware Scheduling With FaMAS," Ieee Transactions on Networking, 2025.doi : 10.1109/ton.2025.3574741.

[7] M. Bahrami, Y.Xu, M.Tweed, B. Bozkaya, and A.S.Pentland, "Using gravity model to make store closing decisions: A data driven approach," Expert Systems with Applications, vol.205, no., pp., 2022. doi:10.1016/j.eswa.2022.117703.

[8] A.Fernandes, L Crespo, N.Neves, P.Tomas, N.Roma, and G.Falcao, "Functional Validation of the RISC-V Unlimited Vector Extension," leee Embedded Systems Letters, vol.17, no.1, pp.2-5, 2025. doi: 10.1109/les.2024.3416820.

[9] M.A.Islam, and K.Kise, "An Efficient Resource Shared RISC-V Multicore Architecture," leice Transactions on Information and Systems, vol. E105D, no. 9, pp. 1506-1515, 2022. doi:10.1587/transinf.2021EDP7248.

[10] C. Busch, M. Herlihy, M. Popovic, and G. Sharma, "Dynamic scheduling in distributed transactional memory," Distributed Computing, vol. 35, no. 1, pp. 19-36, 2022.doi: 10.1007/s00446-021-00410-w.

[11] F. EbrahimiAzandaryani, and D. Fey, "ExTern: Boosting RISC-V core performance using ternary encoding," Microprocessors and Microsystems, vol. 107, no., pp., 2024. doi:10.1016/j.micpro.2024.105058.

[12] Tianzheng Li, Enfang Cui, Yuting Wu, Qian Wei, and Yue Gao, "TeleVM: A lightweight virtual machine for RISC-V architecture," IEEE Computer Architecture Letters, vol. 23, no. 1, pp. 121-124, 2024. doi: 10.1109/LCA.2024.3394835.

[13] Taeho Yoo and Byoung Wook Choi, "Real-time performance benchmarking of RISC-V architecture: Implementation and verification on an EtherCAT-based robotic control system," Electronics, vol. 13, no. 4, pp. 733, 2024. doi:10.3390/electronics13040733.

[14] R. Elkhatib, B. Koziel, R. Azaderakhsh, and M. M. Kermani, "Accelerated RISC-V for Post-Quantum SIKE," leee Transactions on Circuits and Systems I-Regular Papers, vol. 69, no.6, pp.2490-2501, 2022. doi:10.1109/tcsi.2022.3162626.

[15] D. D. Li, and T. M. Aamodt, "Inter-Core Locality Aware Memory Scheduling," Ieee Computer Architecture Letters, vol. 15, no. 1, pp. 25-28, 2016.doi: 10.1109/lca.2015.2435709.

[16] S. Valentin, H. S. Lichte, H. Karl, G. Vivier, S. Simoens, J. Vidal, and A. Agustin, "Cooperative Wireless Networking Beyond Store-and-Forward," Wireless Personal Communications, vol. 48, no. 1, pp. 49-68, 2009. doi: 10.1007/s11277-007-9429-2.

[17] C. Borrego, and S. Robles, "A store-carry-process-and-forward paradigm for intelligent sensor grids," Information Sciences, vol. 222, no., pp. 113-125, 2013. doi: 10.1016/j.ins.2012.08.016.

[18] T. Kimura, T. Matsuda, and T. Takine, "Location-Aware Store-Carry-Forward Routing Based on Node Density Estimation," Ieice Transactions on Communications, vol. E98B, no. 1, pp. 99-106, 2015. doi: 10.1587/transcom.E98.B.99.

[19] M. Tsuru, M. Takai, S. Kaneda, Agussalim, and R. Aina Tsiory, "Towards Practical Store-Carry-Forward Networking: Examples and Issues," Ieice Transactions on Communications, vol. E100B, no. 1, pp. 2-10, 2017. doi: 10.1587/transcom.2016CQI0001.

[20] C. C. Zhao, S. N. Yue, X. Lin, and W. Q. Sun, "Decoupled scheduling in Store-and-Forward OCS networks," Optical Switching and Networking, vol. 35, no., pp., 2020. doi: 10.1016/j.osn.2019.100539.

[21] J. Feliu, A. Ros, M. E. Acacio, and S. Kaxiras, "Speculative inter-thread store-to-load forwarding in SMT architectures," Journal of Parallel and Distributed Computing, vol. 173, no., pp. 94-106, 2023. doi: 10.1016/j.jpdc.2022.11.007.

[22] A. Mihály, and L. Bacsárdi, "Optical transmittance-based store and forward routing in satellite networks," Infocommunications Journal, vol. 15, no. 2, pp. 8-13, 2023. doi: 10.36244/icj.2023.2.2.

[23] K. Stoilova, and T. Stoilov, "Extensions to traffic control modeling store-and-forward," Expert Systems with Applications, vol. 233, no., pp., 2023. doi: 10.1016/j.eswa.2023.120950.

[24] C. J. Li, K. M. Kim, B. C. Wu, P. Z. Zhang, H. Zhang, X. L. Dai, P. Vajda, and Y. Y. Lin, "An Investigation on Hardware-Aware Vision Transformer Scaling," Acm Transactions on

Embedded Computing Systems, vol. 23, no. 3, pp., 2024. doi: 10.1145/3611387.

[25] R. N. M. Watson, D. Chisnall, J. Clarke, B. Davis, N. W. Filardo, B. Laurie, S. W. Moore, P. G. Neumann, A. Richardson, P. Sewell, K. Witaszczyk, and J. Woodruff, "CHERI: Hardware-Enabled C/C plus plus Memory Protection at Scale," Ieee Security & Privacy, vol. 22, no. 4, pp. 50-61, 2024. doi: 10.1109/msec.2024.3396701.

[26] M. Loukil, L. Sfaxi, and R. Robbana, "Scaling down to scale up: benchmarking single-machine graph processing frameworks in a hardware-constrained environment," Computing, vol. 107, no. 7, pp., 2025. doi: 10.1007/s00607-025-01510-2.

[27] B. G. Oripov, A. Dienstfrey, A. N. McCaughan, and S. M. Buckley, "Scaling of hardware-compatible perturbative training algorithms," Apl Machine Learning, vol. 3, no. 2, pp., 2025. doi: 10.1063/5.0258271.

[28] Y. L. Wang, Q. Zhang, C. Nickle, Z. Y. Zhang, A. Leoncini, D. C. Qi, A. Borrini, Y. M. Han, E. del Barco, D. Thompson, and C. A. Nijhuis, "Molecular-scale in-operando reconfigurable electronic hardware," Nanoscale Horizons, vol. 10, no. 2, pp. 349-358, 2025. doi: 10.1039/d4nh00211c.

[29] X. H. Zhang, D. Liu, J. X. Wu, E. P. Cheng, C. Y. Qin, C. S. Gao, L. T. Shan, Y. Zou, Y. Y. Hu, T. L. Guo, and H. P. Chen, "Pixel-Level Hardware Strategy for Large-Scale Convolution Calculation in Neuromorphic Devices," Advanced Functional Materials, vol. 35, no. 17, pp., 2025. doi: 10.1002/adfm.202420045.

[30] V. Jimenez, M. Rodriguez, M.Dominguez, J.Sans, I.Diaz, L Valente, V.L.Guglielmi, J.V.V.Quiroga, R.I.Genovese, N.Sonmez, O.Palomar, and M. Moreto, "Functional Verification of a RISC-V Vector Accelerator," leee Design& Test, vol. 40, no. 3, pp. 36-44, 2023. doi:10.1109/mdat.2022.3226709.

[31] S. Kalapothas, M. Galetakis, G. Flamis, F. Plessas, and P. Kitsos, "A Survey on RISC-V-Based Machine Learning Ecosystem," Information, vol.14, no.2, pp., 2023. doi:10.3390/info14020064.

[32] M. Martonosi, "Embedded systems in the wild: ZebraNet software, hardware, and deployment experiences," Acm Sigplan Notices, vol. 41, no. 7, pp. 1-1, 2006.doi: 10.1145/1159974.1134651.

[33] J. Suhonen, M. Kohvakka, M. Hännikäinen, and T. D. Hämäläinen, "Design, implementation, and experiments on outdoor deployment of wireless sensor network for environmental monitoring," Embedded Computer Systems: Architectures, Modeling, and Simulation, Proceedings, Lecture Notes in Computer Science S. Vassiliadis, S. Wong and T. D. Hamalainen, eds., pp. 109-121, 2006.

[34] R. Wu, and W.-K. Jia, "CSVRF: A CAM-based popularity-aware egress group-caching scheme for SVRF-based packet forward engines," let Communications, vol. 18, no. 1, pp. 40-54, 2024. doi:10.1049/cmu2.12701.