RV-Link: Enhancing RISC-V Multi-Core Efficiency with Hardware Message Passing Channels in a 2D Mesh Network

***Abstract*—Efficient inter-core communication (ICC) is a critical component in multi-core systems, directly impacting performance and scalability. In this paper, we introduce a hardware-software co-designed communication mechanism, referred to as the RISC-V Link (RV-Link), which combines custom hardware and software to enable efficient, low-latency direct message passing in a 2D mesh network. To our best knowledge, RV-Link is the first hardware-based ICC mechanism specifically tailored for RISC-V multi-core systems. RV-Link leverages bi-directional hardware-based queues for communication between neighboring cores, while a complementary subset of Message Passing Interface (MPI) library provides the necessary APIs for seamless interaction between producers and consumers. Unlike traditional software-based message passing techniques that rely on shared memory or interrupts, RV-Link allows exchanging messages with its neighbors, eliminating the need for complex synchronization mechanisms and reducing communication overhead. As a case study, we designed and implemented the proposed RV-Link in the PULP RISC-V-based System-on-Chip (SoC) on a 16nm FinFET process node. Using a range of performance test cases, we demonstrate that RV-Link achieves 6.8–15x lower latency and 6.7–45x higher throughput compared to the software-based MPI approach.**

***Index Terms*—Inter-processor communication, RISC-V, multi-core, MPI.**

I. Introduction

The demand for efficient inter-core communication (ICC) mechanisms has increased significantly with the widespread adoption of embedded multi-core systems in modern computing platforms [1]. ICC is crucial for enabling seamless interaction between cores, directly influencing the performance, scalability, and responsiveness of multi-core architectures. This need is particularly pronounced in embedded applications, such as stream processing, where data flows through multiple cores before producing results. Such workflows pose significant challenges in managing large volumes of data across cores, making latency and throughput highly dependent on the efficiency of inter-core communication. Moreover, as the core count continues to grow, achieving scalable and efficient communication becomes increasingly critical.

Traditionally, shared-memory communication has been the preferred approach due to its straightforward programming model [2]–[4]. However, as the number of cores in modern processors continues to grow, shared-memory techniques face significant scalability challenges. To overcome these limitations, designers of multi-core processors have increasingly adopted message-passing communication, which offers superior scalability and is well-suited for the demands of next-generation embedded multi-core systems [5]–[8]. As the RISC-V ecosystem continues to expand there is a growing need for efficient message-passing architectures tailored to its unique characteristics. Despite advancements in multi-core communication techniques, existing solutions often rely on software-based message-passing mechanisms. These approaches, while flexible, suffer from inherent limitations, including high communication latency, significant synchronization overhead, and suboptimal throughput.

There are numerous architectures where a Network-on-Chip (NoC) is utilized to connect cores; however, its primary role is often limited to linking processing cores with a shared memory controller [9]. When message passing is supported, memory accesses to a shared address space are commonly used for transmitting messages. In this process, the sender writes the message to the receiver's message-passing buffer and notifies the receiver through a separate mechanism that a message has been delivered. The receiver then retrieves the message from its message-passing buffer. These remote read and write operations to the buffers are translated into NoC messages, meaning the actual transmission of messages occurs via the NoC. However, this intermediate translation into memory accesses introduces additional overhead, and the message passing buffers effectively act as a form of shared memory [10]. Depending on the implementation, this can lead to performance bottlenecks. While hardware-based message-passing mechanisms have been explored in other architectures, they remain largely absent in the RISC-V domain, particularly for 2D mesh topologies that are well-suited for scalable and structured inter-core communication.

In this paper, we address these challenges by introducing RISC-V Link (RV-Link), to our best knowledge, the first hardware-based ICC mechanism tailored specifically for RISC-V multi-core systems. RV-Link employs bi-directional hardware queues to enable low-latency, high-throughput message passing between cores, eliminating the need for complex software management mechanisms. A key feature of RV-Link is that it is *privately memory-mapped* to each core, thereby avoiding the overhead typically associated with shared memory communication. This private memory mapping ensures that message passing operations are isolated and efficient, significantly improving performance and scalability. Additionally, RV-Link operates without requiring specific instructions, simplifying its integration and use. We demonstrate the integration of RV-Link into a 2D mesh topology, leveraging its

scalability. Furthermore, RV-Link can be extended to other topologies, such as 3D mesh, torus, hypercube, and tree-based architectures, offering flexibility and adaptability for a wide range of system designs. To complement the hardware design, we propose a subset of the MPI library providing APIs for message exchanges. This co-design of hardware and software ensures tight integration and optimized performance for hardware-based message passing in RISC-V multi-core architectures. The main contributions of this paper are as follows:

1) We propose RV-Link, the first hardware-based memory mapped ICC mechanism designed specifically for RISC-V multi-core systems in a 2D mesh topology, enabling simultaneous message passing between cores and its neighbors for improved communication efficiency.
2) We develop a C-based subset of the MPI library that provides APIs for message exchange, enabling seamless integration between hardware and software.
3) We evaluate RV-Link on a 2D mesh four-core RISC-V System-on-Chip (SoC) based on the PULP platform [15], [16] using a range of performance tests, demonstrating 6.8–15x lower latency and 6.7–45x higher throughput compared to the software-based MPI approach, while introducing a relatively small overhead of only 1.2% in power and 1.06% in area compared to the overall SoC.

The remainder of this paper is organized as follows: Section II provides background and related work, Section III introduces the RV-Link microarchitecture, Section IV presents the MPI library subset developed for RV-Link, Section V describes our experimental results, and Section VI summarizes our conclusions.

II. Background and Related Works

This section provides an overview of *Message Passing*, followed by a discussion of related works.

A. Message Passing

Message passing [5]–[8] is a fundamental communication paradigm in parallel and distributed computing, where data is exchanged between processing units through explicit send and receive operations. It enables coordination and data sharing in systems with multiple cores or nodes, making it essential for scalable and efficient computation. The Message Passing Interface (MPI) [11] is a widely adopted standard that provides a set of APIs for implementing message passing in parallel systems. MPI supports various communication messages, including point-to-point and collective operations, and is traditionally implemented in software as a library. The fundamental operations in message passing include *send* and *receive*, which allow processes to exchange data directly, and *collective operations* such as broadcast, gather, scatter, and reduce, which enable efficient communication across multiple processes. These operations can be categorized into *blocking* and *non-blocking* modes: blocking operations require the process to wait until the communication is complete, while non-blocking operations allow the process to continue computation while the communication progresses in the background, enabling better overlap of communication and computation for improved performance.

B. Prior Works

One of the earliest many-core processors designed for message passing was the Intel Single-chip Cloud Computer (SCC) [12]. The SCC introduced Message Passing Buffers (MPBs), which are small scratchpad memories tightly coupled to each core. These buffers can be accessed by all other cores through the network-on-chip (NoC), facilitating data exchange. To address the significant overhead associated with shared buffer-based message passing, customized instructions for sending and receiving messages at the register level have been proposed in the Sunway SW26010 processor [13]. Building on this concept, PIMP [10] adopts register-level message passing but introduces different instructions to minimize hardware costs in a RISC-V multi-core SoC. Similarly, the Raw microprocessor [5] employs a register-mapped interface designed to work with its underlying packet-switching network protocol. Each core in the Raw processor is equipped with four registers, corresponding to the four independent NoCs integrated into the chip. The Tilera architecture [6] extends this approach by providing the same message passing interface while enabling additional functionality. Specifically, Tilera allows multiple registers to receive messages, supporting up to four concurrent message receptions. Specialized instructions for message passing have also been explored in earlier architectures. The Transputer [14] provides dedicated instructions for sending and receiving messages. Its ‘out‘ instruction is used for sending messages, and the ‘in‘ instruction receives a messages.

The RISC-V ecosystem has not yet fully adopted hardware-based message passing solutions, creating a significant gap in the development of efficient inter-core communication mechanisms for multi-core architectures. This paper addresses this challenge by presenting **RV-Link**, the first hardware-based message passing mechanism designed specifically for RISC-V 2D mesh multi-core systems. The PIMP [10] is the closest work to our study as it also targets hardware message passing for RISC-V multi-core systems. However, there are multiple key differences: (1) it assumes designated instructions to access the message passing hardware queue, while this study employs memory mapping instead, and (2) it assumes send and receive FIFOs connected to a network interface controller that can support various network topologies, whereas we assume a lower cost point-to-point connection between cores and demonstrate it on a 2D mesh network.

III. RV-Link Microarchitecture

The RV-Link microarchitecture, illustrated in Fig. 1 (a), comprises a message-passing queue designed for communication between cores. A single RV-Link instance manages uni-

directional communication between two cores, while achieving bi-directional communication requires deploying two RV-Link instances. It consists of a memory queue, used to transfer message data in 4-byte elements, managed by Enqueue and Dequeue control logic. The Enqueue control manages the queue write address (w_addr), write enable (w_en), and generates a full signal when the queue reaches its maximum capacity. The Dequeue control manages the read address (r_addr), read enable (r_en), and generates an empty signal when the queue is empty. To prevent deadlocks, a watchdog timer is integrated into the design. The timer monitors the message at the head of the queue and, if it remains pending beyond a programmable time threshold, forcibly dequeues the message and sends an error signal to both sender and receiver cores. After dequeuing, the timer resets and resumes monitoring. Additionally, the watchdog timer can be programmed to dequeue multiple consecutive messages or trash all message queue data entries based on a software-configurable parameter, providing further flexibility for handling communication congestion.

The RV-Link design provides configurable mechanisms to handle queue status efficiently during enqueue and dequeue operations. The full and empty status of the queue is indicated through a memory-mapped control register, allowing software to check the queue status before performing operations. Alternatively, the queues can be configured to eliminate the need for software to manually check the status. In this configuration, when an enqueue operation is attempted on a full queue, the control logic will not acknowledge the operation until space becomes available, stalling the core pipeline until the queue is no longer full. Similarly, for dequeue operations, the control logic can be configured to stall the pipeline until data is available in the queue, eliminating the need to check the empty status before proceeding. These optimizations reduce software overhead and streamline inter-core communication.

As a case study, we integrated the RV-Link with the cv32e40p RISC-V core within the Pulpino SoC from the PULP project. The CV32E40P is a compact and efficient 32-bit in-order RISC-V core featuring a 4-stage pipeline. It supports the RV32IMFC instruction set architecture along with PULP custom extensions, enabling improved code density, performance, and energy efficiency [15], [16]. The Pulpino SoC consists of a Tightly Coupled Data Memory (TCDM), an L2 cache memory, a NoC, I/O peripherals and an event unit. Fig. 1(b) illustrates the integration of four RV-Link modules with a core within the Pulpino SoC. The core is connected to SoC memory and peripheral components through the core demux module, which acts as a demultiplexer. To enable efficient communication in a 2D mesh topology, the core demux module has been extended with four additional outbound ports and four additional inbound ports, each connected to dedicated outbound and inbound RV-Link instance respectively. These ports interface with four bidirectional RV-Link modules—RV-Link North (N), South (S), East (E), and West (W)—allowing the core to communicate with its four neighboring cores.

Last, the SoC architecture illustrated in Fig. 1 (c) comprises a core complex with four RISC-V cores connected through RV-Link 2D mesh topology allowing bi-directional message exchange between every core and its four neighbor cores. In addition, the core complex is connected via the NoC to the SoC shared memory (TCDM and L2 cache), DMA, Event Unit and I/O peripherals (such as I2C, UART, and SPI).

IV. MPI Library Subset

To facilitate seamless integration between hardware and software, we develop a lightweight subset of the MPI library tailored for RV-Link. This subset provides essential APIs for message exchange, enabling efficient communication between cores. We support two primary functions: *send* and *receive*, which are specifically designed to interact with the hardware-based communication queues of RV-Link. RV-Link serves as the physical layer, providing the underlying data transfer mechanism, while the MPI functions act as a transport layer carried over this physical layer. This layered approach allows for the addition of further MPI functions in the future, enabling more advanced communication patterns to be seamlessly supported on the RV-Link infrastructure. Below, we describe the *send* and *receive* functions in detail.

A. MPI send()

The *send()* function is used by the software to send a message to one of its neighbors cores in the 2D mesh topology. The function takes the message data, the size of the message, and the destination core ID as arguments. It interacts directly with the hardware queue to push the message for delivery. The *send()* function is non-blocking, meaning the core is not halted while the message is being transmitted to the receiver. Additionally, the function transfers the message header, control information, and the message data itself through the RV-Link message queue, with the data being pushed into the queue in units of 4-byte elements.

```
int send(const void *msg, int s, int dst);
```

- **Parameters:**
  - *msg*: Pointer to the data to be sent.
  - *s*: Size of the message in bytes.
  - *dst*: ID of the destination core (neighboring core in the 2D mesh).
- **Return Value:** Returns 0 on success, or an error code if the operation fails.

B. MPI receive

The *receive()* function is used by a software to receive a message from one of its neighbors. The function takes a buffer to store the incoming message, the size of the buffer, and the source core ID as arguments. It interacts directly with the hardware queue to retrieve the message. The *receive()* function is non-blocking, allowing the core to continue processing while waiting for a message to arrive. The function retrieves the message header, control information, and the message data itself from the RV-Link message queue, with the data being popped from the queue in units of 4-byte elements.

```
int receive(void *buf, int s, int src);
```

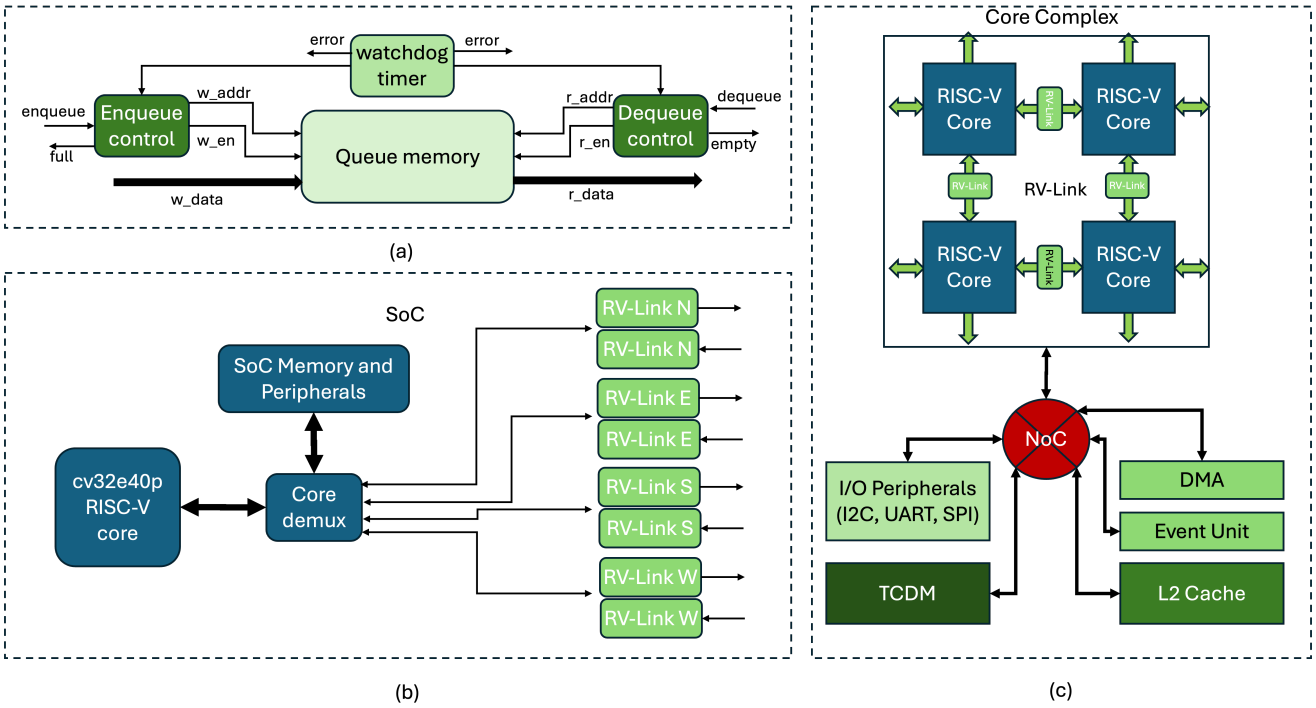

Fig. 1. (a) RV-Link message passing queue microarchitecture, (b) Integration of RV-Link with the RISC-V core, and (c) 2D Mesh RISC-V SoC.

- **Parameters:**
  - *buf*: Pointer to the buffer where the received message will be stored.
  - *s*: Size of the buffer in bytes.
  - *src*: ID of the source core (neighboring core in the 2D mesh).
- **Return Value:** Returns the size of the received message on success, or an error code if the operation fails.

V. Experimental Analysis

This section presents a comprehensive performance analysis, beginning with a detailed description of the experimental and simulation environment, followed by an evaluation of the experimental results, and concluding with an assessment of power and area overheads.

A. Experimental and Simulation Environment

To evaluate the performance of our proposed design, we integrated RV-Link into a four-core RISC-V SoC based on the PULP platform, enabling the formation of a 2D mesh topology. A range of performance tests were utilized to assess the performance of RV-Link and compare it with software-based message passing implemented in shared memory using TCDM or L2 cache. All performance analyses were conducted using System-Verilog simulations. For power analysis, we performed synthesis and place-and-route of the SoC and RV-Link using Cadence Genus and Innovus tools, respectively, targeting a 16nm FinFET process node and assuming a 500MHz clock frequency. The reported area and power figures are derived from post-place-and-route data.

B. Performance Analysis

Our performance analysis includes four tests: (1) unloaded system message latency, (2) unloaded system message latency and throughput, (3) hotspot communication, and (4) all-to-all communication. The details of these tests and their performance results are described below.

*1) **Unloaded System Message Latency**:* We begin our performance analysis by evaluating the latency of transmitting a 32-byte message from one core to another in an unloaded system. Latency is defined as the time taken from when the first 4-byte of the message is sent to when the first 4-byte is received. The results, shown in Fig. 2, compare the latency of RV-Link with software-based message-passing implementations using TCDM and L2 cache. Our analysis reveals that RV-Link achieves the lowest latency of 5 cycles, which is 6.8 times smaller than TCDM (34 cycles) and 15 times smaller than L2 cache-based message passing (75 cycles).

*2) **Unloaded System Message Transfer Time and Throughput**:* Fig. 3 illustrates the unloaded system message transfer time for varying message sizes using RV-Link, TCDM, and L2 cache. The transfer time, measured in clock cycles, is plotted against message sizes ranging from 128 to 4096 bytes. RV-Link consistently demonstrates the lowest transfer time across all message sizes. Specifically, our results show that RV-Link achieves a transfer time approximately 6–9 times lower than TCDM and 24–39 times lower than L2 cache.

Fig. 4 presents the unloaded system throughput measurements for RV-Link, TCDM, and L2. Throughput represents the data transfer rate achieved when continuously streaming

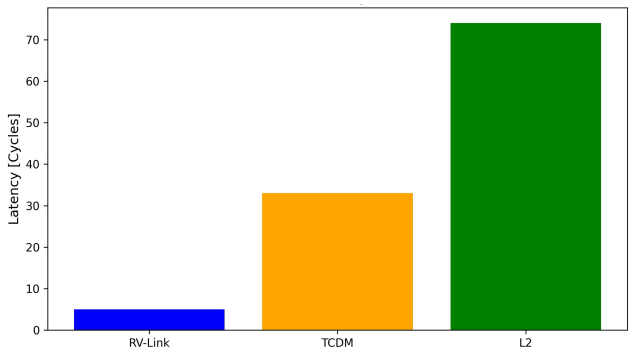

Fig. 2. Unloaded system message latency in clock cycles for RV-Link and software based message passing in shared memory using TCDM and L2 Cache memory.

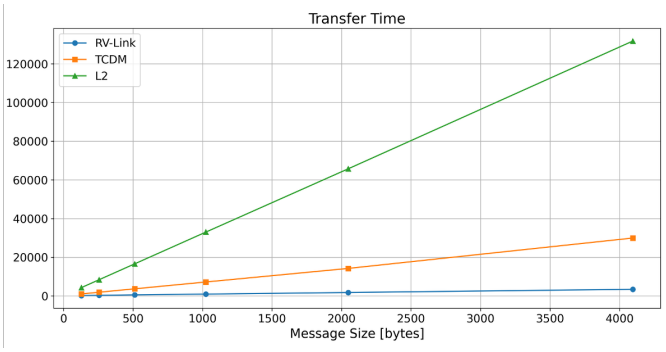

Fig. 3. Unloaded system message transfer time for RV-Link and software based message passing in shared memory using TCDM and L2 Cache memory.

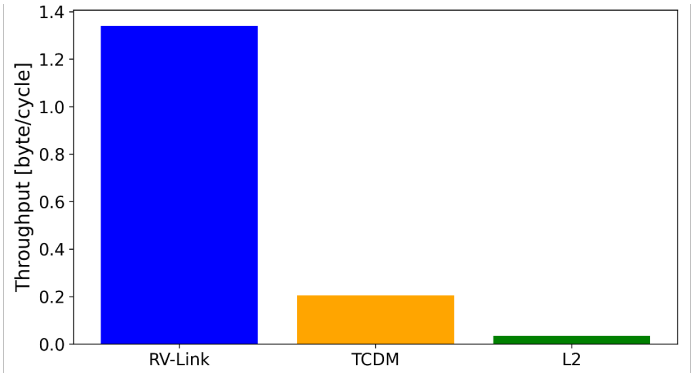

Fig. 4. Unloaded system throughput for RV-Link and software based message passing in shared memory using TCDM and L2 Cache memory.

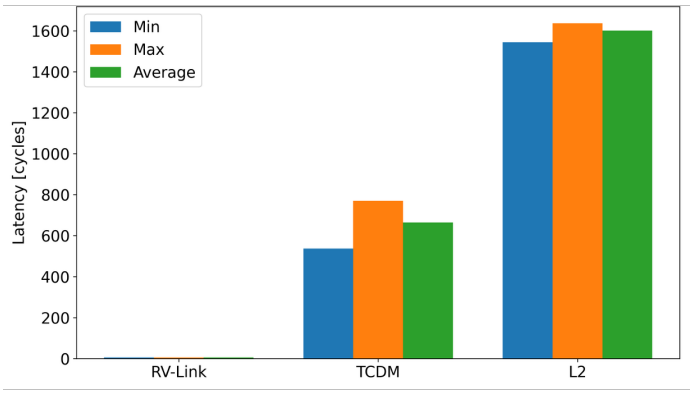

Fig. 5. Hotspot test 32-byte message latency for RV-Link and software based message passing in shared memory using TCDM and L2 Cache memory.

bytes from one core to another in a single direction. RV-Link achieves the highest throughput, exceeding 1.34 Bytes/cycle, significantly outperforming TCDM at 0.2 Bytes/cycle and L2 at 0.03 Bytes/cycle.

*3) **Hotspot Communication**:* The hotspot test evaluates the system's performance under congested communication loads by sending a continuous stream of message data to a single core from all its adjacent cores in the 2D mesh topology. In the case of our four-core SoC, each core has two adjacent cores due to the topology's structure. This test measures two key performance metrics: the variation in message latency and the achieved throughput, providing insights into the system's performance in congested conditions. Figure 5 illustrates the minimum, maximum, and average latency for a 32-byte message across RV-Link, TCDM, and L2 cache. The latency analysis reveals that RV-Link consistently delivers superior performance even in the hotspot test, with a fixed latency of 5 clock cycles. In comparison, TCDM exhibits higher latency, starting at 537 cycles, averaging 664 cycles, and reaching a maximum of 770 cycles. L2, on the other hand, records the highest latency among the evaluated methods, with values ranging from a minimum of 1544 cycles to an average of 1601 cycles and a maximum of 1636 cycles.

Fig. 6 presents the throughput measurements for 32-byte messages between a sender core and a receiver core. The results indicate that RV-Link achieves the highest receiver throughput of 2.68 Bytes/cycle, while TCDM and L2 cache achieve receiver throughputs of 0.08 Bytes/cycle and 0.03 Bytes/cycle, respectively. In all inter-core communication methods, each sender achieves half of the receiver's bandwidth because two senders aggregate their data to a single receiver. An additional observation from this performance test is that the receiver's throughput is twice the value observed in the unloaded system test (Fig. 4), suggesting that the throughput bottleneck lies with the sender rather than the receiver. This is due to the fact that the sender must perform two memory accesses—one to read data from memory and another to enqueue it into the message queue—whereas the receiver only needs to dequeue the data from the queue.

*4) **All-to-All Communication**:* The all-to-all performance test evaluates the system's ability to handle simultaneous bidirectional communication between cores in the 2D mesh topology. In this test, each core continuously sends 32-byte messages to its two neighboring cores while simultaneously receiving messages from the same two neighbors. Unlike the hotspot performance test, where a core operates either as a sender or a receiver, the all-to-all test requires every core to function as both a sender and a receiver, enabling full-duplex communication.

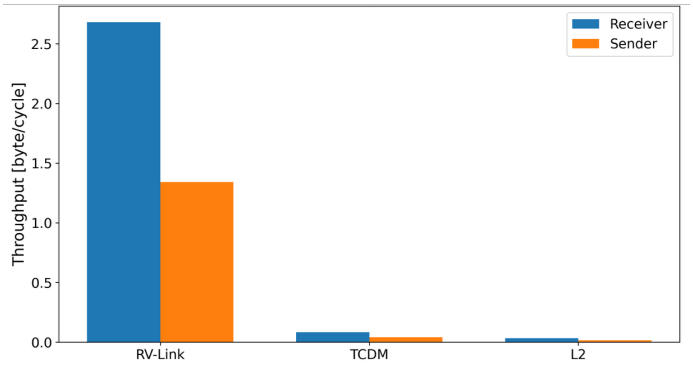

Fig. 6. Hotspot test 32-byte message throughput for RV-Link and software based message passing in shared memory using TCDM and L2 Cache memory.

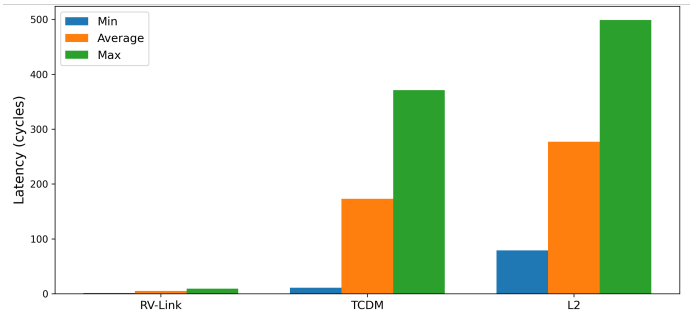

Fig. 7. All-to-all 32-byte message latency test for RV-Link and software based message passing in shared memory using TCDM and L2 Cache memory.

The latency results, depicted in Fig. 7, indicate that RV-Link has the lowest values across all metrics, with a minimum of 4 cycles, an average of 4.79 cycles, and a maximum of 9 cycles. TCDM follows with a minimum of 11 cycles, an average of 173 cycles, and a maximum of 371 cycles. L2 exhibits the highest latency, ranging from a minimum of 79 cycles to an average of 277 cycles and a maximum of 499 cycles. The all-to-all performance test demonstrated lower message latency compared to the hotspot test for several reasons. First, the traffic is distributed evenly among all neighboring cores, which reduces congestion in the message queues. This balanced distribution allows each core to share the communication load, leading to more efficient message handling. Second, in the all-to-all test, cores are required to manage both sending and receiving messages simultaneously, which optimizes their overall message handling capacity. This observation is further supported by our throughput measurements, which will be described next. The throughput measurements, presented in Fig. 8, reveal that RV-Link consistently achieves the highest throughput of 0.73 Byte/cycle across for both sender and receiver cores. TCDM has lower throughput of 0.025 Byte/cycle, while L2 records the lowest throughput at a consistent 0.00144 Byte/cycle.

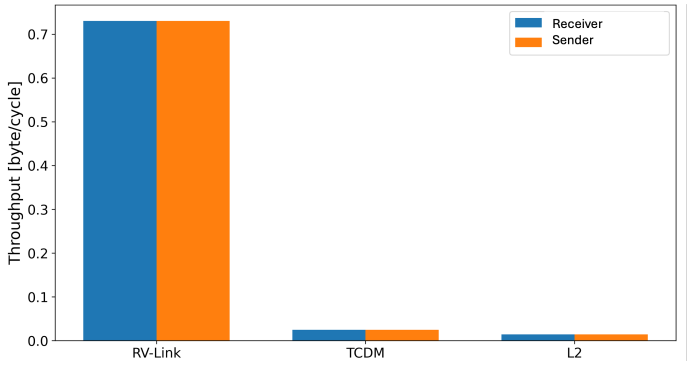

Fig. 8. All-to-all 32-byte message throughput test for RV-Link and software based message passing in shared memory using TCDM and L2 Cache memory.

C. Area and Power Overhead

Table I presents the power and area overhead of the RV-Link integrated to our SoC. The RV-Link consumes 1.77 mW (assuming a Fast-Fast and 0.9V signoff corner) of total power and occupies an area of 0.0037 mm^2, which represents an overhead of 1.2% in power and 1.06% in area compared to the overall SoC, indicating that the power and area overhead of RV-Link is relatively small.

TABLE I
RV-LINK POWER AND AREA OVERHEAD

	Total Power [mW]	***Total Area [mm^2]***
SoC + RV-Link	166.5	0.348
RV-Link	1.77	0.0037
RV-Link Overhead	1.2%	1.06%

VI. CONCLUSIONS

In this work, we have introduced RV-Link, a hardware-software co-designed communication mechanism tailored for efficient inter-core communication in RISC-V-based multi-core systems. RV-Link combines custom hardware and a software API based on a subset of the MPI library to enable low-latency, high-throughput direct message passing between neighboring cores. Through our case study implementation on a 16nm FinFET PULP RISC-V SoC, we have demonstrated the advantages of RV-Link over traditional software-based message passing approaches. Our experimental results show that RV-Link achieves 6.8–15x lower latency and 6.7–45x higher throughput compared to the software-based MPI implementation. Moreover, the power and area overhead of RV-Link is relatively small, at 1.2% and 1.06% respectively, compared to the overall SoC. The RV-Link design represents an important step towards realizing efficient and scalable inter-core communication in future RISC-V-based multi-core systems.

REFERENCES

[1] G. Blake, R. G. Dreslinski, and T. Mudge, "A survey of multicore processors: A review of their common attributes," IEEE Signal Process. Mag., pp. 26–37, Nov. 2009

[2] H.-Y. Kim, Y.-J. Kim, J.-H. Oh, and L.-S. Kim, "A reconfigurable SIMT processor for mobile ray tracing with contention reduction in shared memory," IEEE Trans. Circuits Syst. I, Reg. Papers, no. 60, pt. 4, pp. 938–950, Apr. 2013.

[3] L. Hammond, B.-A. Hubbert, M. Siu, M.-K. Prabhu, M. Chen, and K. Olukolun, "The stanford Hydra CMP," IEEE Micro, vol. 20, no. 2, pp. 71–84, 2000.

[4] A. S. Leon, B. Langley, and L. S. Jinuk, "The UltraSPARC T1 processor: CMT reliability," in Proc. Custom Integrated Circuits Conf. (CICC'06) Dig. Tech. Papers, 2006, pp. 555–562.

[5] M.-B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Stumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw microprocessor: A computational fabric for software circuits and general-purpose programs," IEEE Micro, vol. 22, no. 2, pp. 25–35, Mar/Apr. 2002.

[6] Tilera Corp., Tilepro64 Processor Tilera Product Brief, 2008. Available online: http://www.tilera.com/pdf/Product-Brief_TILEPro64_Web_v2.pdf

[7] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-tile sub-100-W teraFLOPS processor in 65-nm CMOS," IEEE J. Solid-State Circuit, vol. 43, no. 1, pp. 29–41, Jan 2008.

[8] Z. Yu, M. J. Meeuwsen, R. W. Apperson, O. Sattari, M. Lai, J. W. Webb, E. W. Work, D. Truong, T. Mohsenin, and B. M. Baas, "AsAP: An asynchronous array of simple processors," IEEE J. Solid-State Circuits, vol. 43, no. 3, pp. 695–705, Mar. 2008.

[9] Sodani, A., Gramunt, R., Corbal, J., Kim, H.S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.C.: Knights landing: second-generation intel xeon phi product. IEEE Micro 36(2), 34–46 (2016)

[10] Jörg Mische, Martin Frieb, Alexander Stegmeier, and Theo Ungerer. 2019. PIMP My Many-Core: Pipeline-Integrated Message Passing. In Embedded Computer Systems: Architectures, Modeling, and Simulation: 19th International Conference, SAMOS 2019, Samos, Greece, July 7–11, 2019, Proceedings. Springer-Verlag, Berlin, Heidelberg, 199–211. https://doi.org/10.1007/978-3-030-27562-4_14

[11] Clarke, L., Glendinning, I. and Hempel, R., 1994, April. The MPI message passing interface standard. In Programming Environments for Massively Parallel Distributed Systems: Working Conference of the IFIP WG 10.3, April 25–29, 1994 (pp. 213-218). Basel: Birkhäuser Basel.

[12] Mattson, T.G., et al.: The 48-core SCC Processor: the Programmer's View. In: High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–11 (2010)

[13] Zheng, F., Li, H.L., Lv, H., Guo, F., Xu, X.H., Xie, X.H.: Cooperative computing techniques for a deeply fused and heterogeneous many-core processor architecture. J. Comput. Sci. Technol. 30(1), 145–162 (2015)

[14] INMOS Limited: Transputer Instruction set—A Compiler Writer's Guide (1988)

[15] M. Gautschi at al., "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, pp. 2700-2713, Oct. 2017.

[16] P. Davide Schiavone et al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), Thessaloniki, Greece, 2017, pp. 1-8.