2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT) | 978-1-6654-6906-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSICT55466.2022.9963438

A high-performance RISC-V co-processor architecture for fast IP processing

Xinjie Kong[1], Weiliang He[1], Jun Han[1]

[1] State Key Laboratory of ASIC and System, Fudan University, Shanghai 200433, China
Email: junhan@fudan.edu.cn

Abstract
Longest Prefix Matching (LPM) plays an important role in fast IP processing. In this paper, a hardware co-processor architecture connected to high-performance RISC-V processor is proposed to achieve faster LPM search. By applying specific customized instructions to achieve HW/SW co-processing, we can gain 111% performance improvement for every search iteration. The implementation results under TSMC 28nm technology show that the proposed co-processor can work at 1GHz and only require 0.1% extra area cost (5056 um^2) compared to the whole system of the Xuantie processor.

1. Introduction

Higher throughput of Internet processing is demanded due to the development of Internet applications. The critical function of Internet routers is Longest Prefix Matching [1][2]. It will search the routing table according to the destination address of incoming package to find the longest matching entry for the next hop of the routers.

There have been many algorithm-based optimized work to accelerate LPM execution. Some of them [3] manage a two-level direct index table to effectively reduce the memory access frequency, but will occupy huge memory space. Some of them [4] use hash algorithm to compress the IP address to a smaller length for memory accessing, the memory access frequency and memory space is related to hash length. LPM algorithm in DPDK [5] (Data Plane Development Kit) combined above two methods and has been widely used in industry.

The algorithm is effectively optimized through the previous work. However, the execution on CPU is lack of efficiency. The operation like hash algorithm is lack of parallelism due to the redundant control and data dependency. This paper proposed a co-processor architecture which connected to state-of-art open-source RISC-V processor XuantieC910 [6] through the customized co-processor interface to accelerate crucial functions in LPM algorithm. CPU can deliver the customized instructions to co-processor through customized co-processor interface and transfer result to the pipeline.

The rest of paper is organized as follows. Section2 describe the details of our work. Section3 show the experimental results. Section4 make the summary.

2. Proposed Architecture

In this section, we will introduce the proposed architecture in detail. First, we introduce the overall system which run our application. Then, we will focus on the detailed co-processor architecture to accelerate LPM application functions. Finally, we describe the workflow of whole application.

2.1 Overall system

The system consists of three parts: control processor, customized co-processor interface and co-processor. The control processor is based on the state-of-art OOO RISC-V processor XuantieC910. In order to support the execution of customized instruction, customized co-processor interface is proposed to transport the instructions to co-processor and communicate with original pipeline.

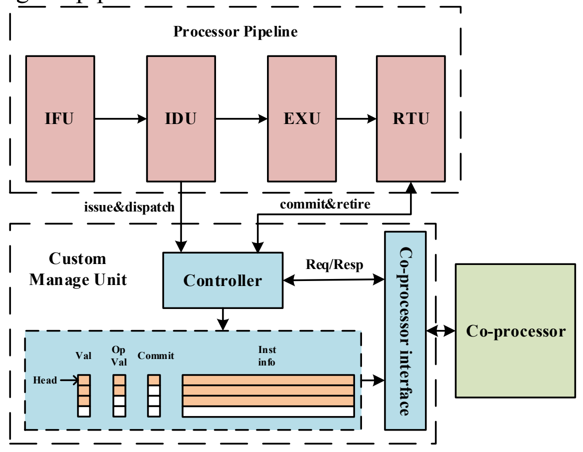

Figure 1. system structure

The whole structure is shown as Figure 1. The pipeline part is based on the XuantieC910, we modify some logic details from original pipeline to support the issue, dispatch and retire behavior of customized instructions. The customized interface contains the Custom Manage Unit (CMU) and co-processor interface which connect to co-processor. The CMU receive the instruction information from IDU and maintain a queue table to manage the behavior of customized instruction. When the customized instruction is committed by processor, co-processor interface will request co-processor for execution. Then co-processor run the instructions and response to interface with the write back result (if have), CMU will transfer it to the retire unit (RTU) for instruction retire.

In this way, co-processor can execute the customized instruction efficiently and communicate with the host processor timely.

2.2 Co-Processor Architecture

Figure 2 shows the proposed co-processor architecture. In order to accelerate LPM algorithm, we first embedded specific hash engine to perform the hash algorithm efficiently. Besides, we implemented Masked Prefix Generator and Prefix Length Searcher to perform prefix preprocessing rapidly. Moreover, we implemented some CSR registers as internal register for data reuse to eliminate duplicated memory access.

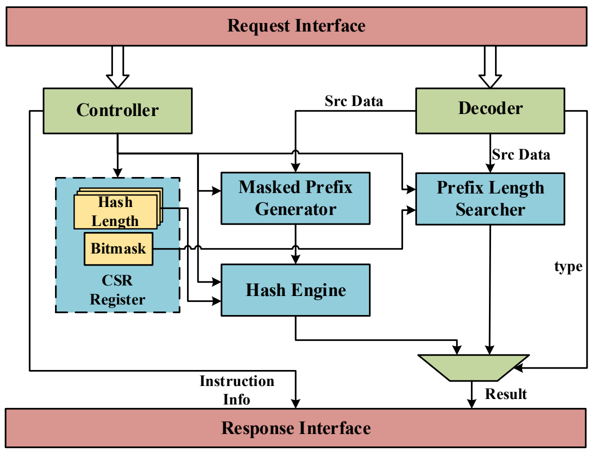

Figure 2. co-processor architecture

We choose fnv-1a algorithm as the hash algorithm of our hash engine. Our engine will return a 32-bit result based on several iterations of multiply and exclusive-or calculation. As the equation 1 shows, every iteration will perform an exclusive-or and a multiply operation. Hash refers to intermediate result, data refers to several bits of input data, and prime is a const value.

$$hash = (hash \oplus data) * prime \qquad (1)$$

As Figure 3 shows, we implemented shift registers and logic gates to realize the calculation of exclusive-or. Besides we change the multiplier to several shift operations and adders to save resources. In addition, a status engine is managed to control the calculation. At the end of iteration, hash engine will mask the 32-bit result based on the value of Hash Length register in Figure 2. Then the final result will be transfer to response interface.

Masked Prefix Generator perform the prefix mask operation based on the given prefix length and IP destination. Prefix Length Searcher calculate the prefix length to be searched in the round based on the value of Bitmask registers in Figure 2.

Moreover, a simple decoder and controller are implemented for dispatching instructions to proper engine and control the signals connected to customized interface.

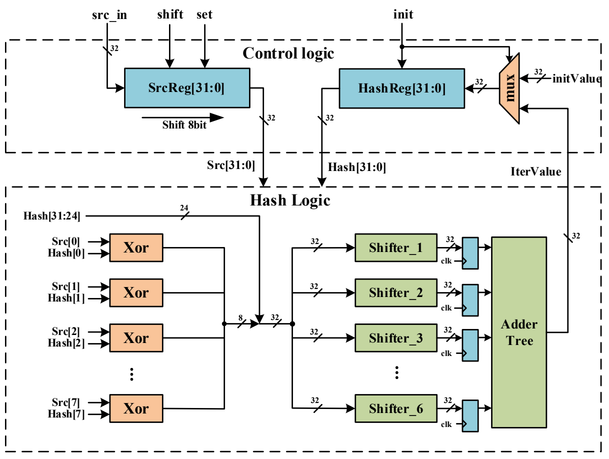

Figure 2. hash engine structure

2.3 Software and Hardware Workflow

Now we can turn to the execution procedure of LPM algorithm. As Figure 4 shows, the LPM search contains a group of comparations. We will calculate the prefix length at the beginning of iteration according to bitmask whose bits indicates the prefix length is used in table. Then, we mask the destination IP and hash it to get the entry address in memory. Then Comparation is performed at the end of iteration.

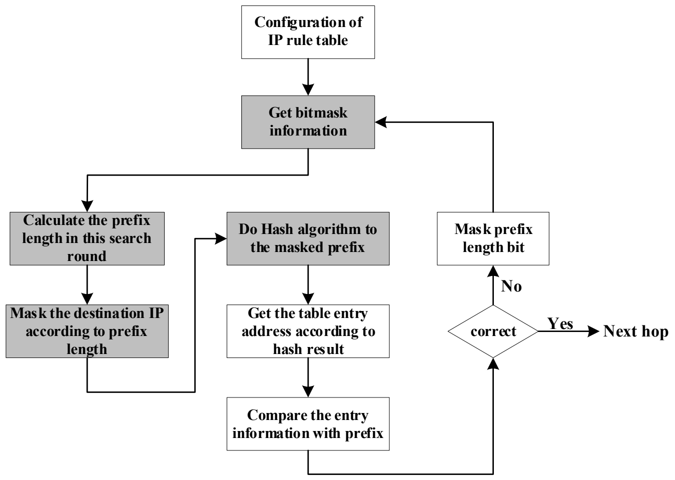
Configuration of IP rule table
Get bitmask information
Calculate the prefix length in this search round
Mask the destination IP according to prefix length
Do Hash algorithm to the masked prefix
Get the table entry address according to hash result
Compare the entry information with prefix
Mask prefix length bit
correct
No
Yes
Next hop

Figure 4. HW/SW workflow for LPM search

Workflows in Figure 4 in gray are performed in co-processor, whereas the rest of workflows are executed by the host processor.

In order to perform the workflow with co-processor, we proposed some customized instructions according to the

RISC-V Specifications [9]. The detailed information is listed in Table1.

Table 1. customized instruction information

Instruction Name	Instruction Coding	Remarks
Set_hashSize	0x2c5b00b	Set hashSize CSR register
Set_bitmask	0x600000b	Set bitmask CSR register
Bitmask_ffs	0x400450b	Find the next prefix length
Hash_Calculate	0x0c5f50b	Preprocessing prefix and calculate hash value

3. Experimental Results and Discussion

The co-processor and customized interface we proposed were implemented into XuantieC910, a high-performance open-source RISC-V processor core developed by T-head. Some modification was made in original pipeline to support our interface.

The system was verified by using the open-source RTL simulator Verilator. The testing program is mainly based on an open-source LPM algorithm library [7]. As Figure5 shows, we gain 84% performance improvement for bitmask processing operation, and 118% performance improvement for prefix preprocessing and hash operations. This is benefit from the good data reuse and calculation parallelism in co-processor. And the overall performance gain is 111% for every searching iteration. Note the sum of separate part CPU cycles is larger than the total cycle, which is due to the pipeline stall caused by rdcycle instruction in RISC-V.

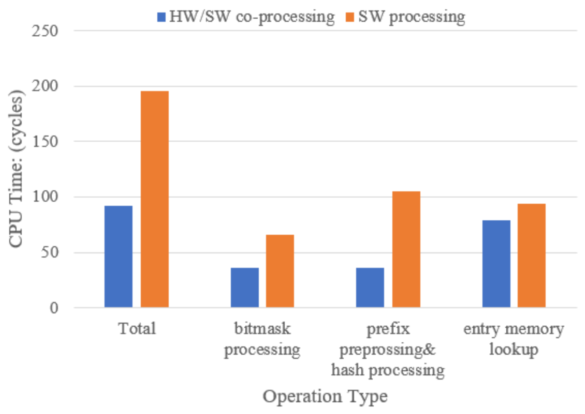

Figure 5. CPU cycles on different operation for SW processing and HW/SW co-processing

Table 3 shows the synthesis result of our co-processor and XuantieC910 core using Design Compiler under TSMC 28nm technology. The Co-Processor only cost 5056um^2 area and 3.92mW power which occupy only around 0.1% in the whole system.

Table 3. synthesis result

	C910 Core	Co-Processor
Frequency (MHz)	1000	1000
Cell Area (um^2)	5175206	5056
Power (mW)	4000	3.92

4. Summary

In this paper, co-processor architecture based on RISC-V processor is proposed for accelerating the LPM algorithm. By cooperating with customized interface and co-processor, the HW/SW co-processing workflow can gain 111% performance improvement for every searching iteration with only 5056 um^2 extra area being used. This design explores the data reuse and calculation parallelism in LPM algorithm, the experimental result indicates this architecture can effectively accelerate the LPM execution.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 61934002.

References

[1] S. S. Ray, A. Chatterjee and S. Ghosh, "A hierarchical high-throughput and low power architecture for longest prefix matching for packet forwarding,", 2013 IEEE International Conference on Computational Intelligence and Computing Research, pp. 1-4 (2013)

[2] S. S. Ray, S. Ghosh and B. Sardar, "SRAM based longest prefix matching approach for multigigabit IP processing," 2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS), pp. 1-6 (2015)

[3] P. Gupta, S. Lin and N. McKeown, "Routing lookups in hardware at memory access speeds," Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 1240-1247 (1998)

[4] S. Ghosh and M. Baliyan, "A hash based architecture of longest prefix matching for fast IP processing," 2016 IEEE Region 10 Conference (TENCON), pp. 228-231 (2016)

[5] "DPDK:Home", https://www.dpdk.org (accessed June. 20th ,2022)

[6] "T-head-Semi/openc910", Github, https://github.com/T-head-Semi/openc910(accessed June. 20th ,2022)

[7] "rmind/liblpm", Github, https://github.com/rmind/liblpm (accessed June. 20th, 2022)

[8] Y. Wang, Z. Qi, H. Dai, H. Wu, K. Lei and B. Liu, "Statistical Optimal Hash-Based Longest Prefix Match," 2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), pp. 153-164 (2017)

[9] "riscv/riscv-isa-manual", Github, https://github.com/riscv/riscv-isa-manual/releases/tag/draft-20220604-4a01cbb (accessed June. 20th, 2022)