Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension

Industrial Product

Chen Chen, Xiaoyan Xiang, Chang Liu, Yunhai Shang, Ren Guo, Dongqi Liu, Yimin Lu, Ziyi Hao, Jiahui Luo, Zhijian Chen, Chunqiang Li, Yu Pu, Jianyi Meng, Xiaolang Yan, Yuan Xie and Xiaoning Qi*

The T-Head Division, Alibaba Cloud
Email: jianyi.mjy@alibaba-inc.com

***Abstract*—The open source RISC-V ISA has been quickly gaining momentum. This paper presents *Xuantie-910*, an industry leading 64-bit high performance embedded RISC-V processor from Alibaba *T-Head* division. It is fully based on the RV64GCV instruction set and it features custom extensions to arithmetic operation, bit manipulation, load and store, TLB and cache operations. It also implements the 0.7.1 stable release of RISC-V vector extension specification for high efficiency vector processing. *Xuantie-910* supports multi-core multi-cluster SMP with cache coherence. Each cluster contains 1 to 4 core(s) capable of booting the Linux operating system. Each single core utilizes the state-of-the-art 12-stage deep pipeline, out-of-order, multi-issue superscalar architecture, achieving a maximum clock frequency of 2.5 GHz in the typical process, voltage and temperature condition in a TSMC 12nm FinFET process technology. Each single core with the vector execution unit costs an area of 0.8 mm^2 (excluding the L2 cache). The toolchain is enhanced significantly to support the vector extension and custom extensions. Through hardware and toolchain co-optimization, to date *Xuantie-910* delivers the highest performance (in terms of IPC, speed, and power efficiency) for a number of industrial control flow and data computing benchmarks, when compared with its predecessors in the RISC-V family. *Xuantie-910* FPGA implementation has been deployed in the data centers of Alibaba Cloud, for application-specific acceleration (e.g., blockchain transaction). The ASIC deployment at low-cost SoC applications, such as IoT endpoints and edge computing, is planned to facilitate Alibaba's end-to-end and cloud-to-edge computing infrastructure.**

***Index Terms*—RISC-V, multi-core, cache, memory architectures, out of order, vector, extension**

I. Introduction

Cloud computing and IoT applications are fueling the wave of semiconductor research and development. The demand for low-power and cost-effective CPUs is ever increasing. The RISC-V is very attractive at this point in time, because: *i*) as an alternative to closed and costly ISAs, the open and free ISA of RISC-V accelerates processor innovation through open standard collaboration; *ii*) its scalability, extensibility, and modularity enable processor customization and optimization for domain-specific workload (e.g., machine learning accelerators, network processing, security enclave, storage controllers and supercomputing), thereby boosting processing efficiency and reducing design cost; *iii*) RISC-V is becoming a mainline platform in Unix/Linux OS. The toolchains like GNU/GCC/GDB and LLVM are getting mature, further improving software experience and driving down software development cost.

Admittedly, compared with the X86, ARM, MIPS [16]–[18], PowerPC, SPARC [11], [20], [21], [23], [30], openRISC [14], [24], [26] and other ISAs under the hood of popular GPUs and DSPs, RISC-V is still in its infancy. We see many RISC-V efforts have been making inroads, some of which are highlighted below:

- Academic work - The UC Berkeley released the in-order *Rocket* core, the out-of-order core *BOOM* and the open-source design generator tool [9], [10]. Both *Rocket* and *BOOM* are capable of booting Linux. ETH Zurich and Università di Bologna offered three flavors of RISC-V cores in the *PULP* platform [4] - *RI5CY* (32-bit, 4-stage pipeline), *Zero-riscy* (32-bit, 2-stage pipeline) and *Ariane* (64-bit, 6-stage pipeline) [32]. Also, IIT-Madras has been working on a series of *Shakti* RISC-V processors, from a 3-stage pipeline in-order core to an out-of-order multiple threading core at a target operating frequency of 1.5-2.5 GHz [5], [13].
- Industrial work - For many years nVIDIA has been using RISC-V as the *Falcon* controllers in the GPUs [8]. Besides, chip vendors such as SiFive, Microsemi, Alibaba T-Head, Andes, Codasip offer a range of silicon-proven 32-bit and 64-bit embedded RISC-V IPs. Recently, Western Digital open-sourced the SweRV^{TM} Core [25], which is an industry-quality 32-bit, 2-way superscalar, 9-stage pipeline core. This power-efficient design reaches over 1.0 GHz operating frequency in a TSMC 28nm CMOS process technology. With a performance of up to 5.0 CoreMarks/MHz (based on internal simulations)

This paper is part of the Industry Track of ISCA 2020's program.

and small footprint, it offers compelling capabilities for embedded devices for data-intensive edge applications, such as storage controllers, industrial IoT, real-time analytics in surveillance systems and other smart systems.

Along the RISC-V performance spectrum, most of the existing cores are in the microcontroller class [12], [15], [19], [33]. Some prior arts extended RISC-V to domain-specific accelerators/coprocessors [22], [27]–[29]. However, as of today very few options are available at the 64-bit high performance end.

The *Xuantie* product series from Alibaba T-Head are high-performance embedded computing cores based on RISC-V architecture. The T-head Semiconductor (or Pingtouge Semiconductor) is the business entity of Alibaba Group specializing in IC chip designs, with the primary goal of developing the next-generation of cloud integrated chip architecture, data centers and embedded IoT chip products. The codename *Xuantie* refers to a ”heavy sword made from dark iron” in Chinese tales. The intention of this work is NOT to compete with any non-RISC-V cores on the market, but rather contribute to the high-end 64-bit RISC-V architecture through open-source collaboration. To date, only the FPGA implementation has been deployed in the data centers of Alibaba Cloud, for application-specific acceleration by taking advantage of the custom extensions and the vector extensions. The initial FPGA deployment scale is limited to several hundreds. The implementation on the Xilinx VU9P FPGA runs at 200 MHz frequency in Linux OS. Taking blockchain transaction acceleration as an example, in terms of per-core performance, the FPGA edition is still 20% higher than the x86_64 Intel Xeon Platinum 8163 CPU that runs at 2.5 GHz in ubuntu16.04 OS. A cost-down ASIC edition has been taped out and the chip is expected in July, 2020. It is projected to run at a frequency of 2.0 - 2.5 GHz, resulting in 12-15X higher performance than the x86_64 Intel Xeon Platinum 8163 CPU counterpart. In addition to internal use, Alibaba has been promoting *Xuantie* core IP series to facilitate external customers for edge computing applications, such as AI, edge servers, industrial control and advanced driver-assistance systems (ADAS). Alibaba is also in the preparation process of open-source release of *Xuantie*. By the end of 2022, a total volume of 15 million units is expected. The full product portfolio can be viewed at Alibaba T-head semiconductor division’s website at https://www.t-head.cn/product.

Prior to our work, the SiFive *U74* core [6] was the world’s highest performance RISC-V application processor. It is capable of achieving up to 5.1 CoreMark/MHz, an accomplishment on par with the ARM Cortex-A55 core [1]. *XT-910* (abbr. for *Xuantie-910*) is a 12nm 64-bit RISC-V processor with 16 cores clocked at up to 2.5 GHz frequency. It can perform out-of-order execution and has a triple-issue 12-stage pipeline. *XT-910* implements the RV64GCV, meaning that it supports the base 64-bit RISC-V ISA (RV64G) and supports compact 16-bit-wide instructions (C) as well as the standard 32-bit wide instructions. It features RISC-V vector extension. It also includes more than 50 non-standard instructions to accelerate various tasks. The toolchain has also been largely optimized to improve the efficiency and performance. *XT-910* processor reaches 7.1 CoreMark/MHz, which is 40% faster than *U74*.

The remainder of this paper is organized as follows. Section II gives an overview to the *XT-910* architecture. Section III and section IV introduce the instruction fetch unit and the execution core, respectively. Section V discusses the memory sub-system in detail, from the hardware implementation to the memory management in Linux OS. Section VI briefly goes over the multi-cluster and multi-core SMP. Section VII and VIII present the vector instruction extension and non-standard instruction extensions that boost the performance of domain-specific applications. Section IX shows the optimized compilation tool chain. The experimental results on various benchmarks are provided in section X. Finally, section XI draws conclusion of the paper.

II. *XT-910* Architecture Overview

XT-910 is fully in compliance with the RISC-V RV64GCV instruction set specification [31]. It supports the standard user (U), supervisor (S), and machine (M) privilege modes, as shown in Fig. 1. The RV64GCV designation means that *XT-910* implements *i*) the base 64-bit RISC-V ISA (RV64G); *ii*) compact 16-bit-wide instructions (C) as well as the regular 32-bit-wide instructions, and *iii*) the vector operations (V), whose standard is still in development. In addition, *XT-910* has more than 50 non-standard instructions to accelerate various domain-specific tasks.

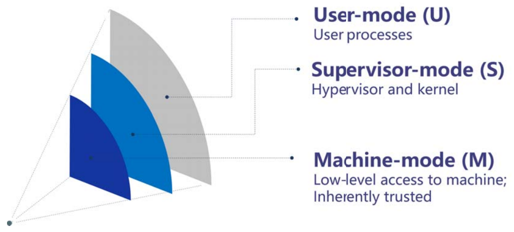


Fig. 1: The U, S, M privilege modes.

XT-910 exploits a homogeneous multi-cluster architecture. As shown in Fig. 2, a cluster is composed of up to 4 cores. Each core supports a 32/64 KB L1 instruction cache and a 32/64 KB data cache. Each cluster has a shared, inclusive 8/16-way associated L2 cache, which has up to 8 MB and supports both ECC and parity check. Exclusive memory access instructions are also supported. *XT-910* includes a standard 8-16 region PMP (Physical Memory Protection) and a SV39 MMU, which is compatible with the RISC-V Linux specification. The LSU supports unaligned memory data access. It also incorporates standard CLint and PLIC multi-core interrupt controllers, timers and debuggers, performance monitors and I/O slave interfaces. The supported core configurations are listed in Table I.

To improve the performance, *XT-910* features RISC-V vector extension and a series of non-standard custom extensions including extensions to arithmetic, bit manipulation, load & store, and TLB & cache operations. In addition, there are

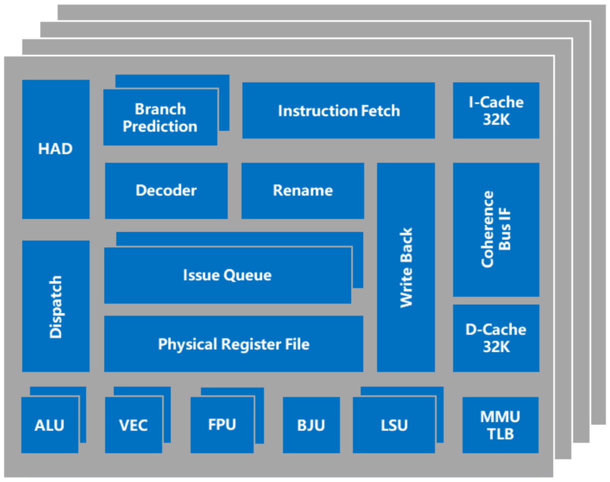


Fig. 2: The diagram of the *XT-910* multi-core cluster (4-core configuration).

TABLE I: *XT-910* core configurations.

feature	configuration
Core Number per Cluster	1, 2, 4
L1 Data Cache	32KB, 64KB
L1 Instruction Cache	32KB, 64KB
L2 Cache Size	256KB $\sim$ 8MB
Vector Extension	yes / no

extensions for the MMU to support page-based memory attribute management and for the interrupt controller to support permission control. All the extensions are supported by our compilation toolchains. Through hardware configurations, the non-standard extensions can be disabled, allowing *XT-910* to operate in a mode fully compatible with the standard RISC-V.

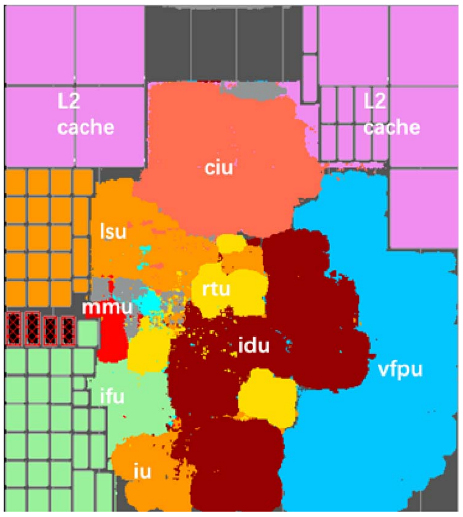
Fig. 3 shows the exemplar floorplan of a single-core and dual-cores. The performance from post-layout simulation is summarized in Table II. In a TSMC 12nm FinFET process technology, the implemented silicon areas are 0.8 mm^2 and 0.6 mm^2 for a single-core with and without the vector execution unit, excluding the L2 cache. When operating at 0.8V Vdd, the core reaches over 2.0 GHz frequency using LVT standard cell library and ULVT SRAMs. If we use ULVT standard cells and operate at 1.0V Vdd (i.e., voltage boosting mode), a 2.5 GHz main frequency can be achieved. In another experiment with a 7nm FinFET technology, the frequency of a single core can reach 2.8 GHz.

TABLE II: *XT-910* core performance in a 12nm FinFET.

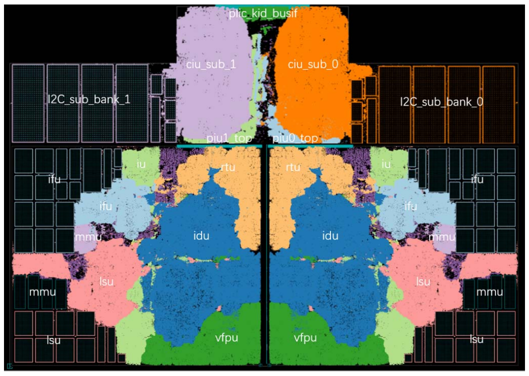
Operating frequency	2.0 GHz[a] $\sim$ 2.5 GHz[b] (TT 85°)
Silicon area per core	0.6 (without VEC) / 0.8 (with VEC) mm^2
Dynamic power	$\sim$100 μW/MHz per core[c] (TT 85°)

[a] Use LVT 6T-turbo standard cell, ULVT SRAM, 0.8V Vdd
[b] Use 30% ULVT standard cell, ULVT SRAM, 1.0V Vdd
[c] Configuration: 32/64KB L1$, 256/512KB L2$, without VEC

The pipeline of the *XT-910* core is shown in Fig. 4. The ”frontend” of the pipeline consists of 7 stages (i.e., IF $\sim$



(a) The layout of a single-core, with vector execution unit and 512KB L2 cache.



(b) The layout of dual-cores.

Fig. 3: The layout of a single-core and dual-cores.

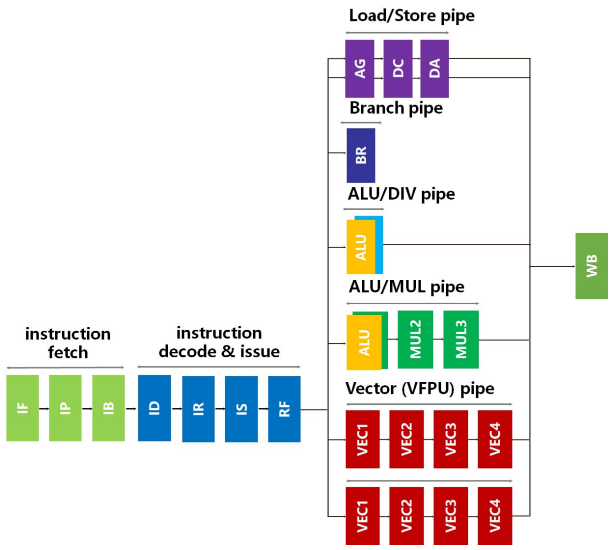


Fig. 4: 12-stage pipeline in *XT-910* core.

RF). The instruction fetch unit (IFU) uses a hybrid predictor, which makes predictions of branch direction, branch address, function return address and indirect jump address. The instruction decoding unit (IDU) can decode 3 instructions simultaneously and can rename up to 4 instructions using physical registers. The out-of-order issue engine can issue up to 8 instructions. The ”backend” of the pipeline has multiple execution units, including two single-cycle ALUs, one single-cycle branch jump unit, one dual-issue out-of-order load & store unit, two scalar floating point units and two vector execution units. The multi-cycle ALU unit and the division unit share one pipe; the integer multiplication units and the two single-cycle ALUs share a pipe. The LSU supports unaligned memory data access, and supports multi-mode multi-stream prefetching to increase data access bandwidth. More details about the IFU, the execution core, the memory subsystem and the memory management unit (MMU) will be discussed in the following sessions.

III. Instruction Fetch Unit

As illustrated in Fig. 5, the IFU in *XT-910* is divided into three pipeline stages:

- Instruction Fetch (IF): access L1 instruction cache to obtain instructions, convert virtual addresses to physical addresses and process the first-stage branch jump. A maximum of 128 bits of instruction line can be fetched from the L1 cache in a single cycle.
- Instruction Pack (IP): pack and pre-process the instructions, process the second-stage of branch jump, and refill lines in case of cache miss. Up to 8 instructions can be packed. As the subsequent IDU can maximally decode 3 instructions, the throughput of IP will not become the performance bottleneck.
- Instruction Buffer (IB): the packed instruction is buffered; up to 3 instructions are sent to the ID pipeline stage, and the third-stage of jump is processed.

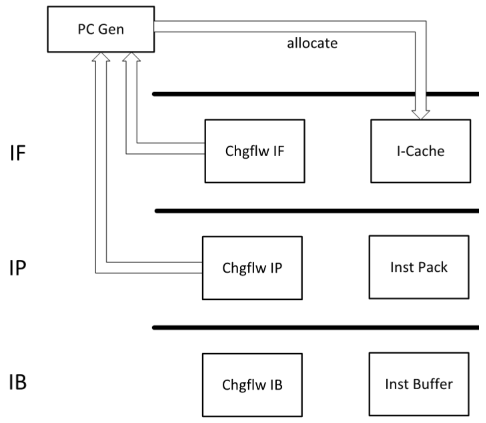


Fig. 5: The *XT-910* IFU pipeline.

XT-910 uses the *hybrid multi-mode branch prediction* to predict the directions and target addresses of branches, including absolute branch, conditional branch, indirect branch, and function call return. This allows program jumps to be initiated as early as possible, reducing pipeline bubbles caused by program flow changes. All instructions prefetched through branch predictions are cached in the Instruction Buffer (IBUF). When the branch prediction is correct, the IBUF content is accumulated. Therefore, even when cache miss occurs, IFU can still provide sufficient instructions to the subsequent IDU pipeline stage.

A. Branch Direction Prediction

To improve the instruction fetch efficiency, the IFU strives to predict the jump direction of conditional branch at the earliest possible pipeline stages. At the IF stage, a small number of branches are captured and jumps are processed right away, so for these branches the delay penalty is reduced to zero. Most of the conditional branches are captured at the IP stage. The predication is based on the branch history. The prediction outcome is stored in multiple memory banks, from which one data is selected to be final result by a dynamic monitoring algorithm. To be able to store a large number of prediction values, the banks are realized in high-density SRAMs.

Because the IF stage has a line width of 128 bits (containing a maximum of 8 instructions), it is highly likely that the instructions have conditional branch instructions in each cycle and the jump of a branch is influenced by the direction of its previous branch jump. However, due to the large access latency of SRAM banks, when a memory read happens, the output value from the SRAM banks (that store the branch prediction values) must be buffered in the registers first before being consumed. As a result, after receiving the predicted value of jump direction for the current branch instruction, one clock cycle delay occurs if this value is used as the historical information for the subsequent branch instruction. In other words, conditional branch instructions at two adjacent cycles cannot be processed consecutively. To resolve this issue, *XT-910* adopts a branch prediction mechanism based on prefetching from multi-level buffers. The possible prediction values of the conditional branch instructions are read out in a fuzzy matching pattern and then buffered in BUF1 and BUF2. When a branch is detected, the value in BUF1 is used for prediction. The associated value from BUF2 moves up to BUF1 to predict the branch at the next cycle, as shown in Fig. 6.

This prediction mechanism can also predict multiple branch instructions in a single cycle. If the 128-bit instruction fetched in one cycle contains multiple branch instructions, the prediction result of the first branch instruction can be obtained from BUF1. Based on the result, the prediction result of the second branch instruction can be obtained from BUF2. With a higher level of prefetch buffers, more branch instructions can be predicted at the same time. When misprediction happens, the error will be corrected only when the branch instruction reaches the branch jump execution unit, which causes at least seven clock cycles of performance penalty compared to executing jump at the IP stage.

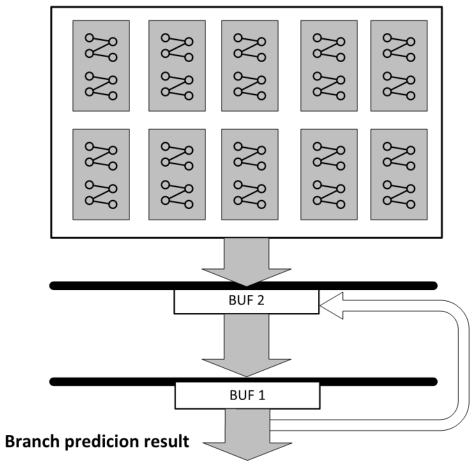


Fig. 6: The branch predictor with 2-level buffer.

B. Branch Target Prediction

The branch predictor only predicts if a branch instruction requires to jump, but the prediction of the jump address is also important. Even if a jump is correctly initiated at the IP stage, a bubble still occurs in the pipeline. When jumps occur too frequently, the bandwidth of the IFU cannot meet the demand of the subsequent execution units. To eliminate the pipeline bubble, the IFU adopts a cascaded branch target buffer (BTB). The L1 BTB is the main BTB. It has more than 1K table entries and uses set-associative structure. The L0 BTB is fully-associative with 16 table entries. At the IF stage, if the branch instruction hits the L0 BTB, the predicted address stored in the table will be used as the jump address to execute a jump immediately, eliminating the bubble in the pipeline.

The prediction result of the L0 BTB (including the jump direction and the target address) is confirmed at the IP stage. The jump direction is checked by the branch predictor, and the target address is checked by the L1 BTB. In case of L0 BTB misprediction, the target address will be corrected right away at the IP stage.

Conventionally, the branch prediction result of L1 BTB is corrected at the late stages of the pipeline. In *XT-910*, the IFU checks the L1 BTB target jump address at the IB pipeline stage. If the L1 BTB generates a misprediction, an immediate correction is performed.

In most cases, the performance loss caused by a jump instruction at the IP stage will be hidden by the prefetched instructions in IBUF. The L0 BTB is intended primarily for programs whose performance loss cannot be hidden. For instance, if a program with continuous jumps performs a jump at the IP stage, it will frequently generate pipeline bubbles. The number of instructions in the IBUF will be reduced and the bubbles cannot be hidden. The L0 BTB captures program of this kind and caches the branches in it. It initiates jumps at the IF stage to remove the bubbles in the pipeline. Besides, the IFU also has an indirect branch predictor for indirect branch instructions. The subroutines call the return stack to predict the return addresses.

C. Loop Buffer

To facilitate instruction fetch for small loops in programs, the IFU also has a loop buffer (LBUF), as illustrated in Fig. 7. Software program tends to have loop bodies which contain a small number of instructions. When executing loops, jumps occur frequently. If a jump instruction is performed at the IP stage, many bubbles will be inserted. Moreover, the last instruction of the current loop cannot be issued together with the first instruction of the next loop, simply because there is a jump backwards instruction in between.

The LBUF largely solves such issue. The loop body is entirely buffered in the LBUF. The last instruction of the current loop can be issued together with the first instruction of the next loop to the subsequent pipeline stage. In this way, the IFU ensures that the maximal number of instructions (i.e., 3 instructions/cycle) are sent to the IDU. The loop body allows forward branches, so the loop body with *if-else* structure can be also accelerated. Another advantage of the LBUF is that the instruction fetch for loop cases does not require accessing the L1 instruction cache, thereby reducing the power consumption.

XT-910's loop buffer has 16 entries. When a context switch occurs, the loop buffer is flushed.

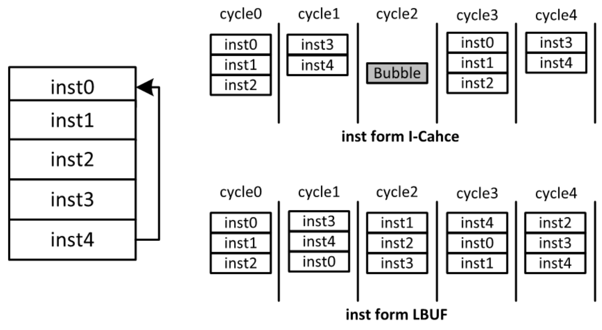


Fig. 7: The loop buffer (LBUF) accelerates program with small loops.

IV. Execution Core

The execution core includes ID, IR, IS, RF, EX1$\sim$EX4, RT1$\sim$RT2 pipeline stages. The IDU is from the ID to the RF stages. The ID stage decomposes and decodes instructions. It splits instruction into micro-instructions based on the attributes (e.g., instruction type, number of operands, and number of write-back operations, etc.). Zero-latency decoupling of scalar and vector instructions is supported.

The IR stage renames operands in GPR, FGPR and VGPR using physical registers. Register renaming is applied to scalar integer, floating point and vector registers. Register renaming not only effectively handles data dependencies, it also eliminates the use of the costly *move* instruction. *XT-910* adopts speculative allocation of physical registers. Upon the completion of write-back, an instruction retires and the physical registers are released.

The IS stage performs out-of-order instruction scheduling. The processor has 8 instruction slots, which are shared by all the issued instructions. Depending on the available resources and workload of the execution unit, multiple independent out-of-order issue queues are loaded. The issue queues use an *Age-Vector* based scheduling algorithm to issue instructions to the shared instruction slots. In addition, the issue queues also support dynamic load balancing by monitoring the workload in the pipeline and adjusting the priority in real time.

The EX stage contains 8 pipes, which can process 2 arithmetic operation instructions, 1 branch instruction, 1 load instruction, 2 store instructions (i.e., the pseudo double store instructions, as will be discussed later), 2 scalar floating point and vector instructions in parallel.

The RT stage takes care of instruction write-back and retirement. The RTU (retirement unit) retires instructions and releases the physical registers. To ensure the correctness of program execution, the instructions are retired in order in spite of the out-of-order execution. Like many high-performance processors [7], this feature is realized through the re-order buffer (ROB). The ROB can hold up to 192 instructions. When an exception occurs on one instruction, the instruction has to retire and the speculative executions of the following instructions in the pipe are also flushed, as illustrated in Fig. 8.

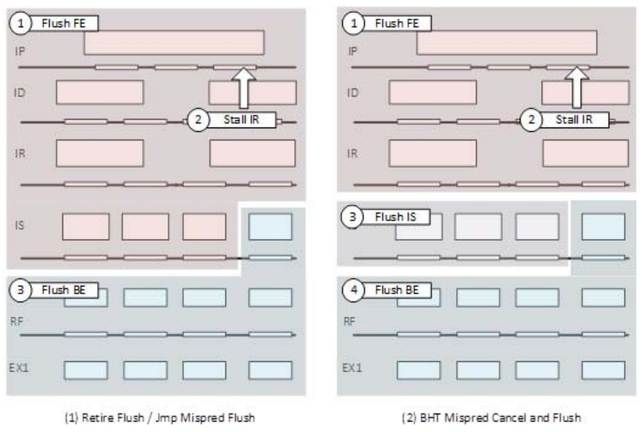

1 Flush FE
IP
ID
2 Stall IR
IR
IS
3 Flush IS
3 Flush BE
4 Flush BE
RF
EX1
(1) Retire Flush / Jmp Mispred Flush
(2) BHT Mispred Cancel and Flush

Fig. 8: Speculative failure recovery after instruction execution exception.

V. Memory Sub-system

Many efforts have been made to enhance the efficiency of *XT-910*'s memory sub-system, as discussed below.

A. Dual-issue out-of-order LSU

To date, *XT-910* is the only RISC-V processor that support dual-issue out-of-order LSU. The LSU can process one load instruction and one store instruction in parallel. The dual-issue LSU requires dedicated load pipe and store pipe. Each pipe contains four pipeline stages, namely address generation (AG), data cache (DC), data alignment (DA), and write-back (WB), as shown in Fig. 9. At the AG stage, the load and store instructions generate addresses, access the uTLB, and convert virtual addresses to physical addresses. At the DC stage, load and store instructions access the data cache. At the DA stage, data alignment is completed. Finally, the data is written back to the physical register file at the WB stage.

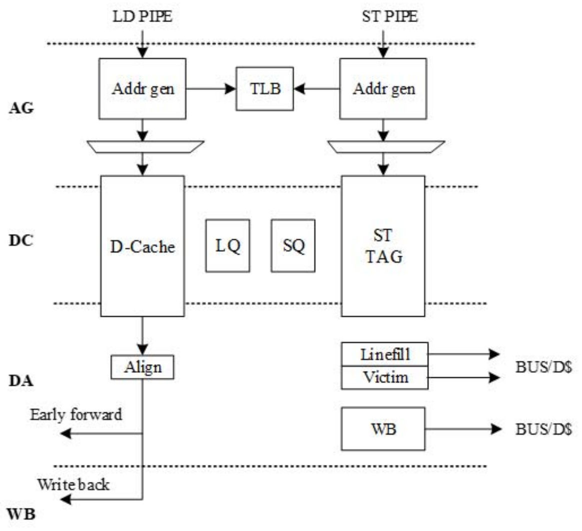


Fig. 9: The pipeline stages in the LSU.

The data cache system includes the DATA RAM and the LD TAG RAM for the load pipe. An additional ST TAG RAM is added to the store pipe. Executions on the load and store pipes access their own RAMs. For proper out-of-order LSU execution, the Load Queue (LQ) and the Store Queue (SQ) are implemented to keep the order of instructions and check the address dependencies. When executing a load instruction, all prior store instructions in the SQ are checked, and the store instructions with the same address are forwarded. When executing a store instruction, the LQ is checked. Once a "younger" load instruction with the same address is to be executed earlier, the speculative execution fails and a global flush is generated. To improve the efficiency of memory access speculation and reduce the performance loss caused by speculative failures, *XT-910* predicts and tags load/store instructions that may cause speculative execution failures, so the execution is blocked by the execution unit to ensure that the load instruction is not executed ahead of the store instruction.

B. Pseudo double store instructions

To accelerate the execution of store instruction, before being issued to the queue, the instruction is decomposed into two micro-operations (μOps), namely the *st. addr* (which stores the address portion of the store instruction, for address generation and cache query) and *st. data* (which stores the data portion of the store instruction, for fetching operands). The *st. addr* goes from the shared load-store issue queue to the store pipe. *st. data* goes from its dedicated *st. data* issue queue to the *st. data* pipe, accesses the physicial register file and forwards the results from the execution units. *st. addr* and *st. data* are merged in the write buffer (WB) in the SQ after going through their own pipes and are written back to the data cache or the off-chip storage (see Fig. 10). By separating the address and the data stores, the address generation and cache access can be performed earlier.

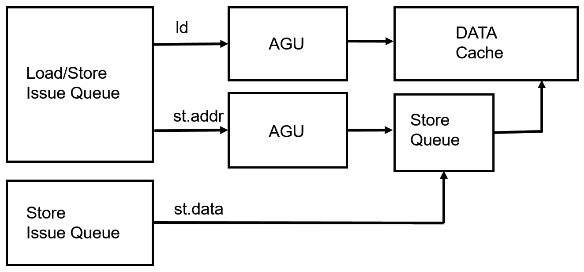


Fig. 10: Independent address and data flows in the LSU.

C. Multi-mode multi-stream data prefetch

For high-performance applications, there exists a large gap between the memory access bandwidth/latency and the processor running speed. Although multi-level cache can dwarf the impact of memory wall, there are still many scenarios where either data is used very few times or the cache capacity is insufficient.

XT-910 provide a unique multi-mode and multi-stream data prefetch approach to enhance the data prefetch capability. By pattern matching the data streams, *XT-910* effectively prefetches data and backfills the L1 or the L2 cache, and reduces the probability of processor stall caused by high memory latency. Two data prefetch modes are supported, as shown in Fig. 11. One is the global prefetch mode, which is suitable for simple but continuous data stream. This mode supports any stride lengths, and the maximum depth of the prefetch is 64 cache lines. The other is the multi-stream prefetch mode, which is suitable for complex scenarios. This mode supports up to 8 data streams with different stride lengths, and supports prefetch depth of up to 32 cache lines.

The prefetch operation is divided into three steps. The first step is stride length calculation - a process which finds the correct pattern from the load instruction addresses. The second step is prefetch control. It decides the prefetch policy and evaluate the confidence. Prefetch policy is responsible for setting the depth of prefetch and dynamically adjusting the start and stop of prefetch. It prevents overly aggressive prefetch from contaminating the cache, or overly slow prefetch from performance degradation. Confidence evaluation determines whether the prefetch policy being used is accurate or whether it needs to be modified or abandoned. The last step is the actual execution of data prefetch. *XT-910* supports multi-stream data prefetch to reduce the performance loss caused by memory latency. In addition, it supports virtual address cross-page prefetch. When data is prefetched at the page boundary, a conversion for the next virtual page is automatically requested.

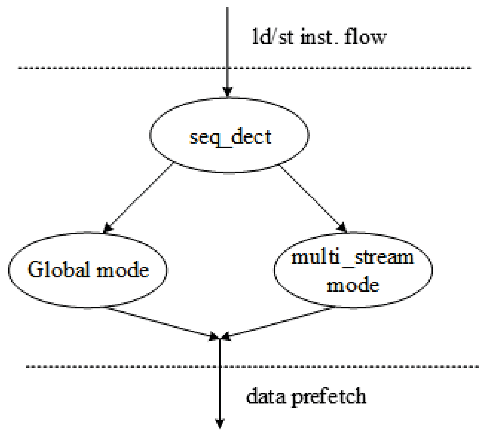


Fig. 11: Multi-mode multi-stream data prefetch.

Once the physical page address is obtained, data prefetch resumes.

D. Multi-size multi-level TLBs

The *XT-910* core supports multi-size (4K, 2M and 1G) entry at all levels of TLBs. Each entry of the micro-TLB and the joint-TLB contains a page-size property. When the micro-TLB (fully-associated) is accessed, the requested virtual address (*va*) is compared with the virtual page number (VPN) in every micro-TLB entry. If the micro-TLB misses, the missed *va* is sent to the joint TLB (jTLB) to find a valid entry.

As illustrated in Fig. 12, the jTLB is 4-way set associative and can only be accessed by one type of index at one time. It can be accessed by three kinds of indexes correspondent to the three different page sizes. It is accessed by the 4K index firstly, and then the 2M index if the 4K request misses, and then the 1G index if 2M request misses. The corresponding entry of jTLB is refilled to micro-TLB on page hit. When all page size requests are missed, page-table walk is triggered.

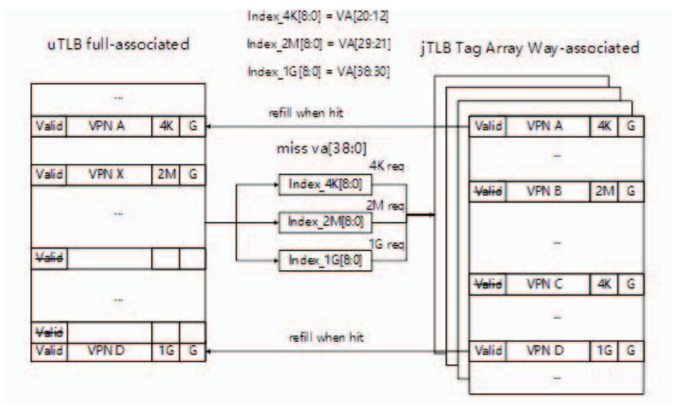


Fig. 12: Multi-size multi-level TLBs.

E. Memory management in Linux OS

To meet the memory management requirements in Linux OS, *XT-910* adds multiple instructions to improve the consistency of MMUs/cache: *i)* It can specify the ASID/base address of page table/virtual address, and can broadcast TLB maintenance information through the interconnection bus. The CPU cores and other peripheral IPs on the interconnection bus can parse the information to maintain their own MMUs. Compared with the IPI (Inter-Processor Interrupt) scheme, the maintenance is performed by hardware without software intervention, hence improving the efficiency. *ii)* It supports huge page mapping [3], which is an important feature required by Linux OS to reduce TLB miss rate. The MMU provides 3 levels table mapping. Each level can be mapped as a leaf table entry. It can simultaneously meet the Linux 4KB, 2MB, 1GB different size of huge page requirements. *iii)* Its ASID width is increased to 16-bit. The TLB needs to be flushed when ASID overflows. The test result shows that, the number of TLB flushes caused by context switch is decreased by almost 10X, as the 16-bit-wide ASID takes a much longer time to overflow.

VI. The multi-core architecture

XT-910 adopts the SMP multi-core architecture. Up to 4 cores are grouped into one CPU cluster and up to 4 CPU clusters are connected using *Ncore* (see Fig. 13). The cores are connected through an internal bus with coherence protocol. They share an inclusive L2 cache with a maximum configurable size of 8 MB. The L2 cache supports MOSEI coherence protocol. A snoop filter that monitors access by the cores to the shared L2 cache effectively reduces the inter-core communications.

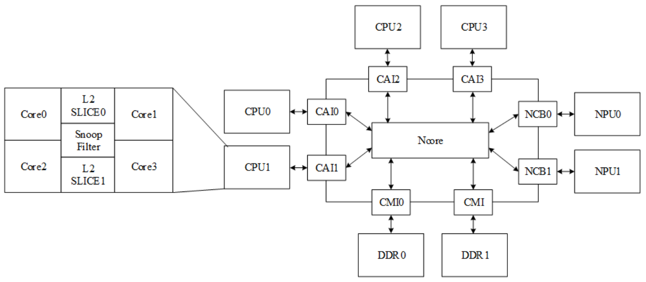


Fig. 13: The block diagram of multi-core implementation.

VII. Vector Instruction Extension

XT-910 is one of the first commercial processors that adopt the RISC-V Vector Extension proposal. The vector engine significantly boosts the performance of vector processing, especially for AI and machine learning applications.

Compared with traditional fixed-length SIMD instruction sets (e.g., ARM NEON and Intel SSE/AVX), variable-length vector instruction set (e.g., ARM SVE and RISC-V Vector Extension) not only provide more flexible parameter selection for hardware, but also enhances the portability of upper-layer software across different hardware platforms. Based on the 0.7.1 stable release version of RISC-V Vector specification, *XT-910* supports the execution of dual-issue out-of-order vector operation instructions.

XT-910 uses a 64-bit scalar pipeline. According to the specification, the vector pipeline supports 8-bit to 64-bit vector integer operations, and half-precision/single-precision/double-precision floating-point vector operations. Its vector pipeline consists of multiple identical vector slices. Each vector slice has a complete 64-bit data path, including a multi-port 64-bit vector physical register file and two out-of-order vector floating-point and integer execution pipelines. Each pipeline supports a single 64-bit integer or double-precision floating-point operations, or two 32-bit integer or single-precision floating-point operations. Each vector slice has its independent registers, forwarding path, and execution data path. Only very few operations, such as *wide*, *narrow*, and *permutation*, need to exchange data across slices.

Fig. 14 shows the vector slices based architecture, which greatly reduces the layout and routing costs caused by the increasing bit-width. *XT-910* can support the operation width from 64 bits to 1024 bits. However, for deeply pipelined out-of-order multi-core processors, the maximum bit-width of load/store access is largely limited by the bus and cache architecture. Further, a higher bit-width increases the cost of memory access and exacerbates cache coherence problem. To balance the bit width ratio of arithmetic-logic operations to load/store access, two vector slices with 128-bit VLEN and SLEN are recommended. With this configuration, *XT-910* can generate a total of 256-bit operation results in one clock cycle, and complete a 128-bit vector load/store operation.

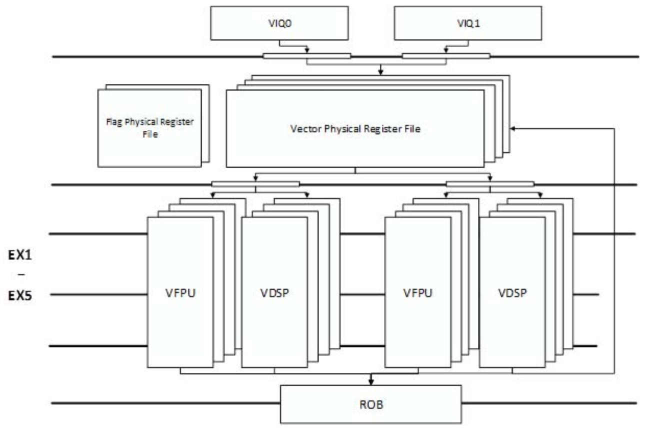


Fig. 14: The pipelined vector operation architecture.

The vector operation instruction itself does not specify the element format and operation bit width. These vector parameters are set through a configuration instruction (*vsetvl/vsetvli*). The parameter configuration instruction only needs to specify the number of elements to be processed, and the hardware can determine the specific operation width and number of elements according to VLMAX. This feature ensures that software does not need to know the underlying hardware parameters but can run on hardware platforms with different computing bitwidths. However, this is not friendly to deeply pipelined processor architecture. The vector parameter correlation has to be inferred from each vector operation instruction and its preceding parameter configuration instruction, slowing down the execution of vector operation instruction. To alleviate the performance loss, *XT-910* enables vector parameter prediction and speculative execution of vector operation, which only cause speculation failures when the *vl* changes.

Most vector operations can be completed within 3-4 clock cycles. Multiplying single and double precision floating point vectors takes 5 clock cycles. Integer division and floating-point division take 6 to 25 clock cycles.

VIII. Non-standard Instruction Set Extension

Targeting at various industrial applications, *XT-910* enables a set of custom non-standard instructions, additional to the standard RISC-V instructions. The non-standard instruction extension can be categorized into two groups based on the purposes - *memory access enhancement*, and *basic arithmetic operation enhancement*.

A. Memory access enhancement

Memory access instructions usually account for a high proportion in the total number of instructions, so enhancing memory access instructions can directly benefit the overall performance. By analyzing the mainstream applications running on RISC-V, we observed that for the basic RISC-V instructions there is still quite some room for improvement in memory access related instructions.

First, we support *register + register* addressing mode, and support indexed load and store instructions. This type of instruction extension reduces the usage of the registers for calculation and reduces the number of instructions for address generation, thereby effectively accelerating the data access of a loop body. Second, unsigned extension during address generation is supported. Otherwise, the basic instruction set does not support direct unsigned extension from 32-bit data to 64-bit data, resulting in too many shift instructions.

B. Arithmetic operation enhancement

Domain-specific applications, such as the security encryption and audio/video codec, present very different arithmetic requirements. For example, security encryption algorithms need to perform frequent *shift*, *and*, *or* and other operations on certain bytes or sections in the data. Therefore, a series of bit operation instructions, multiplication instructions, and multiplication and accumulation instructions have been extended.

IX. The optimized tool chain

We release the *Comprehensive Integrated Development Environment (CDS)* for *XT-910*, which is an IDE that enables RISC-V graphical trace, profiling and instruction accurate simulator with JTAG online debug. GNU's official compilation tool collection is fully supported, as shown in Fig. 15. A snapshot of the profiling tool is shown in Fig. 16.

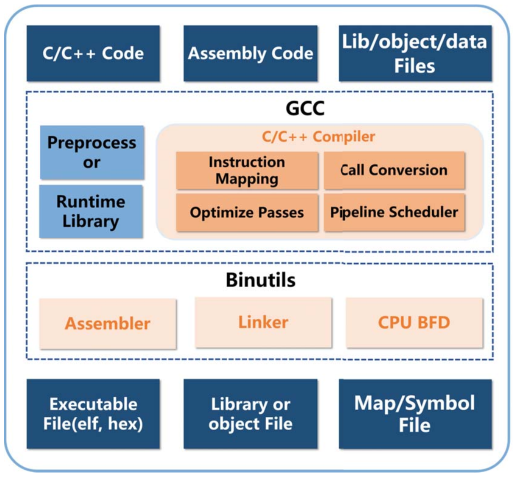
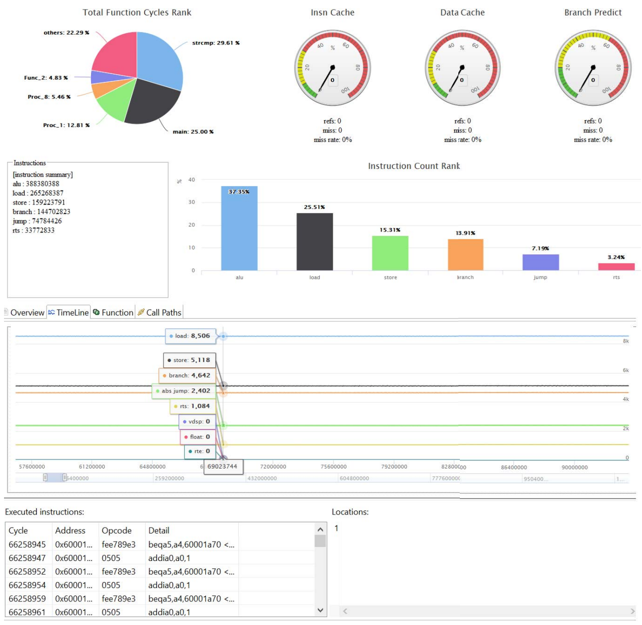


Fig. 15: The compilation toolchain.



Instructions
[instruction summary]
alu : 388380388
load : 265268387
store : 159223791
branch : 144702823
jump : 74784426
rts : 33772833

Executed instructions:

Cycle	Address	Opcode	Detail
66258945	0x60001...	fee789e3	beqa5,a4,60001a70 <...
66258947	0x60001...	0505	addia0,a0,1
66258952	0x60001...	fee789e3	beqa5,a4,60001a70 <...
66258954	0x60001...	0505	addia0,a0,1
66258959	0x60001...	fee789e3	beqa5,a4,60001a70 <...
66258961	0x60001...	0505	addia0,a0,1

Locations:
1

Fig. 16: The *XT-910* profiling tool.

The compiler of *XT-910* has been co-optimized along with the hardware architecture, including:

1) RISC-V architecture uses a signed extension when converting 32-bit to 64-bit data type. Because the existing compilers do not support induction variable optimization, index auto-increment and branch jumps occur during loop unrolling, largely deteriorating the efficiency. The *XT-910* compiler tool extracts the loop variables and moves the auto-increment variable and the control code out of the loop, hence reducing the total number of the instructions that needs to be executed.

2) RISC-V architecture designates the global pointer register (global pointer) to point to a fixed memory address during program execution, allowing some variables to be directly accessed using the load/store instructions by adding an offset to the global pointer register value. Usually, to improve the usage of the global pointer, the compiler will allocate relatively small variables to the regions that can be accessed by global pointer. However, when accessing a small variable and a large variable consecutively, the addresses of these two accesses can be far apart (i.e., large address offset), resulting in poor data locality. Our compiler tool deploys an *anchor* scheme, which allocates the variables of the same function to a continuous address space, saves the starting address of this space to a register and accesses the variables by adding offset to the register content.
3) While the existing RISC-V compilers do not support Dead Store Elimination (DSE) optimization, *XT-910* compiler tool does support DSE.

X. Experimental Results

A series of industrial benchmarks for embedded CPUs have been performed to evaluate the performance. One of the benchmarks is the *Coremark*, which contains implementations of the following algorithms: list processing (find and sort), matrix manipulation (common matrix operations), state machine (determine if an input stream contains valid numbers), and CRC (cyclic redundancy check). Such benchmark is only for core performance evaluation, as it is basically all cache-hit and it is hardly affected by DDR latency. In Fig. 17, the performance is compared with various embedded processors. *XT-910* processor reaches 7.1 CoreMark/MHz, which is 40% faster than *SiFive U74* (which is by far the highest performance RISC-V processor available on market). We saw the news about the upcoming *SiFive U84* processor, but because there are no official product data by the time submitting this article, we cannot make comparison without knowing the evaluation conditions, hardware configuration and software compilation details. We would be happy to include it as a new reference point once the information becomes available.

Because there are hardly benchmarking results available from high performance RISC-V processors of this kind, we use the ARM Cortex-A73 [2] as our reference. As high-performance embedded processors, Cortex-A73 and *XT-910* have many architectural similarities (e.g., pipeline stages, instruction issue width, etc.). This allows us to better analyze and compare the advantages and disadvantages of RISC-V and ARM at the ISA level. The experimental Cortex-A73 processor is from the *Huawei Kirin-970*, which uses the *linaro GCC 6.3-2017.02* toolchain. It has 64KB L1 instruction cache, 64KB L1 data cache, and 2MB L2 cache shared by the instruction and data. For a fair comparison, *XT-910* is configured for the same L1 & L2 cache sizes and it uses *GCC 8.1* compilation toolchain with *-O2* option. **A disclaimer that the benchmarking results are by no means suggesting *XT-910* is up to the perfection of the flagship Cortex-A73 core**. *XT-910* has just made its debut, a multi-year open-collaboration is needed to cover numerous corner cases.

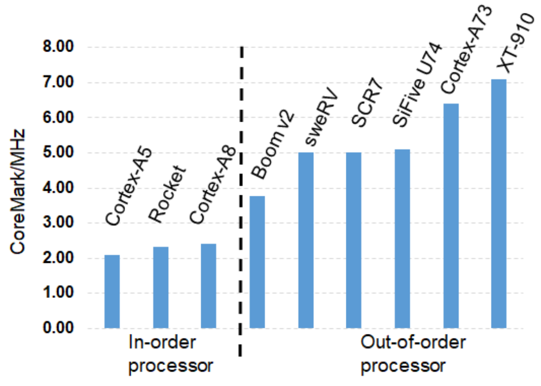


Fig. 17: The CoreMark Scores.

Fig. 18 shows the performance of the EEMBC benchmark, which is a benchmark for the hardware and software used in autonomous driving, the Internet of Things, mobile devices, and other applications.

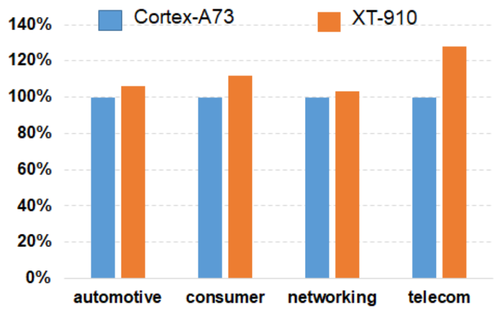


Fig. 18: The performance of the EEMBC benchmark, normalized to the performance of ARM Cortex-A73.

Similarly, we evaluated the *NBench*, which is a performance testing framework for .NET applications. The results are shown in Fig. 19. Overall, the performance of *XT-910* is on par with the ARM Cortex-A73.

We have run the *SPECInt2006* benchmark. *SPECInt2006* uses very large programs that frequently incur L2 cache misses. It factors in core performance, cache size, cache miss, DDR latency, etc., thereby providing a comprehensive evaluation on a CPU system. The performance of *XT-910* is 6.11 SPECInt/GHz, which is 10% lower than the 6.75 SPECInt/GHz delivered by Cortex-A73.

Compared with the native RISC-V ISA and compiler, the performance of *XT-910* with instruction extensions and optimized compiler has been improved by about 20%, as shown in Fig. 20. Although we do not show the acceleration of AI use cases in this paper, it should be noted that *XT-910* supports the RISC-V 0.7.1 vector extension and the Cortex-A73 uses ARM's NEON technology. Taking 16-bit multiply-

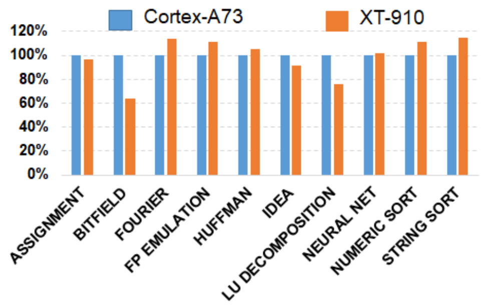


Fig. 19: The performance of the NBench, normalized to the performance of ARM Cortex-A73.

accumulate as an example, as far as we know, the Cortex-A73 supports 8X 16-bit-MAC operation, and the computing power of *XT-910* is 16X 16-bit MACs, so theoretically *XT-910* has a 1X performance improvement over the Cortex-A73. In addition, *XT-910* supports half-precision operation (which is not supported by Cortex-A73), further widening the performance gap between them in AI scenarios.

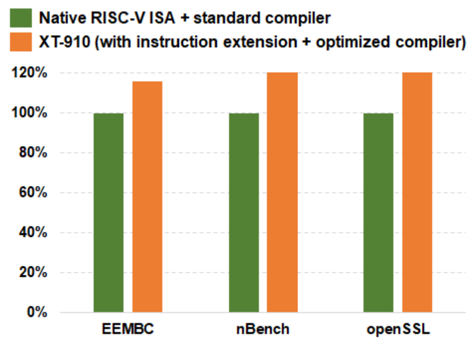


Fig. 20: The performance of *XT-910* with instruction extensions and optimized compiler, normalized to using native RISC-V ISA and compiler.

Fig. 21 illustrates the impact of data pre-fetch on *stream* for the memory subsystem. *stream* is a set of benchmark that tests memory access performance and prefetch performance. The performance is normalized to the performance of scenario *a)*, where all prefetches on off. Scenario *b)* only enabled L1 prefetching, the performance increased to 3.8X. Scenario *c)* further turned on L2 and TLB prefetching, the performance improvement goes up to 4.9X. When the prefetch distance was adjusted from small to large in scenario *d)*, the performance reached a maximum of 5.4X. Scenario *e)* turns off TLB prefetching while other configurations remain the same as scenario *d)*, and the performance is slightly reduced (by about 2.4%). The experimental data are all from evaluation on HAPS80-S26 FPGA. Because the access delay of memory data directly affects the test results, the memory access delay is adjusted to about 200 CPU clock cycles (by specifying the bus delay and DDR delay), i.e., the CPU issues a read request and obtains the data from the bus after 200 CPU cycles. In the FPGA test, the CPU is running at 60MHz.

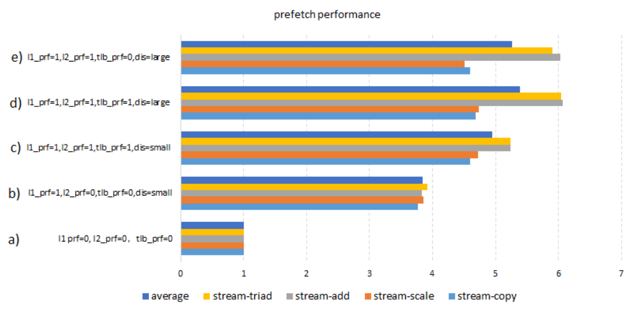


Fig. 21: The performance impact of prefetch on the memory sub-system. *a)* All prefetches are off; *b)* L1 prefetch is on, L2 and TLB prefetches are off, and small distance is configured; *c)* L1, L2, and TLB prefetches are on, and small distance is configured; *d)* L1, L2, and TLB prefetches are on, and large distance is configured; *e)* L1 and L2 prefetches are on, TLB prefetch is off, and large distance is configured.

XI. Conclusion

This paper presents *XT-910* - a deeply pipelined out-of-order high performance 64-bit embedded multi-cluster multi-core RISC-V processor. It is fully based on the RV64GCV instruction set and features custom extensions to enhance vector and arithmetic operations, bit manipulation, load & store, and TLB & cache operations. *XT-910* supports multi-core multi-cluster with cache coherence. Each cluster contains 1 to 4 core(s) capable of booting Linux operating system. Each single core utilizes the state-of-the-art 12-stage pipeline, out-of-order, multi-issue superscalar architecture, attaining a maximum clock frequency of up to 2.5 GHz in the typical process and temperature condition in a TSMC 12nm FinFET process technology. Each single core costs an area of 0.8 mm^2 and 0.6 mm^2, with and without the vector execution unit (excluding the L2 cache). The toolchain of *XT-910* also has been largely optimized to support the custom extensions. By hardware and software co-design, to date *XT-910* delivers the highest performance (in terms of IPC, speed and power efficiency) for a number of industrial control flow and data computing benchmarks when compared with its predecessors in the RISC-V family.

Since the official product launch in July, 2019, the FPGA implementation of *XT-910* has been already deployed in the data centers of Alibaba Cloud, for application-specific acceleration by offloading tasks from servers to FPGAs and taking advantage of *XT-910*'s custom extensions and vector extensions. An ASIC edition has been taped out and the ASIC deployment will happen after silicon bring-up. Besides internal use, the IP has been adopted by dozens of external customers in edge and IoT endpoint computing applications. We are currently enhancing features like multi-core expansion, parallel computing and functional safety. Compared with the

ISA of ARM, the ISA of RISC-V is architecturally more concise. This reduces the decoding and execution costs of processor hardware. It also reduces the complexity of the tool chain. RISC-V ISA is a modular instruction set. Through the combination of different subsets, the processor can be more flexible to obtain advantages in power, area and performance in different application scenarios. While RISC-V ISA's flexibility and scalability provide unlimited possibilities for open innovations, many excellent features in ARM (such as bitwise operation, trace and virtualization) have not yet been finalized in RISC-V ISA proposals.

The R&D journey of *XT-910* convinced the RISC-V community that, *i)* as an open source architecture, the RISC-V architecture not only enables low-power low-cost embedded MCUs, it is also applicable to high-performance computing. For a variety of general-purpose applications, the RV64GC-based *XT-910* has reached the performance of commercial cores in mainstream architectures; *ii)* the RISC-V vector extension could also be well suited for AI and machine learning applications; *iii)* the potential of RISC-V will not only be limited to the edge computing. For many applications in data centers, its flexibility and customizability are beneficial, especially being the scalable domain-specific accelerators in the post-Moore's Law era.

Charles Darwin said, "It is not the strongest of the species that survives, nor the most intelligent. It is the one that is most responsive to change." We believe the philosophy also applies to computing architectures. Nonetheless, the evolving RISC-V architecture is not mature enough in terms of technology and ecosystems. We have been actively participating in the RISC-V foundation on the standardization and discussing the effective custom extensions of *XT-910* with the technical subgroups. Some of the extensions (such as cache operations) have already drawn attention and are considered into future RISC-V standard ISA release. For RISC-V architectures to be more competitive as compared to other industry's mainstream high-performance architectures, the RISC-V community must maintain the ISA conformity and innovate together through software and hardware co-design.

Acknowledgement

We thank the CPU hardware and the software engineering teams in *Alibaba T-Head* division for their tremendous efforts. We also thank *Alibaba Cloud Infrastructure & Service* division for the strong support.

References

[1] "Arm cpu cortex-a55," https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a55?utm_source=google&utm_medium=cpc&utm_campaign=2019_brand_mk30_cpus_search_bol&utm_source=google&utm_medium=cpc&gclid=EAIaIQobChMI6ujPg_3V4wIVWB-tBh3migfFEAAYASAAEgL9MvD_BwE.

[2] "Arm cpu cortex-a73," https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a73?utm_source=google&utm_medium=cpc&utm_campaign=2019_brand_mk30_cpus_search_bol&utm_source=google&utm_medium=cpc&gclid=EAIaIQobChMIioGT1Z_u4wIVB9lkCh2zYQr7EAAYASAAEgKjiPD_BwE.

[3] "Performance tuning guide - red hat enterprise linux," https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/ch-intro#idm140449023557936.

[4] "Pulp platform," https://pulp-platform.org/.

[5] "Shakti processor program," https://shakti.org.in/.

[6] "The sifive u74 standard core," https://www.sifive.com/cores/u74.

[7] "Intel 64 and ia-32 architectures optimization reference manual," Intel Corporation, Tech. Rep., June 2011. [Online]. Available: http://www.cs.princeton.edu/courses/archive/spr15/cos217/reading/ia32opt.pdf

[8] "Risc-v in nvidia," in *6th RISC-V Workshop, Shanghai*, 2017.

[9] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[10] C. Celio, D. A. Patterson, and K. Asanović, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, Jun 2015. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html

[11] J. Feehrer, S. Jairath, P. Loewenstein, R. Sivaramakrishnan, D. Smentek, S. Turullols, and A. Vahidsafa, "The Oracle Sparc T5 16-Core Processor Scales to Eight Sockets," *IEEE Micro*, vol. 33, no. 2, pp. 48–57, Mar. 2013. [Online]. Available: http://ieeexplore.ieee.org/document/6493301/

[12] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V soc for AI at the edge of the iot," in *29th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2018, Milano, Italy, July 10-12, 2018*, 2018, pp. 1–4. [Online]. Available: https://doi.org/10.1109/ASAP.2018.8445101

[13] N. Gala, A. Menon, R. Bodduna, G. S. Madhusudan, and V. Kamakoti, "SHAKTI processors: An open-source hardware initiative," in *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID 2016, Kolkata, India, January 4-8, 2016*, 2016, pp. 7–8. [Online]. Available: https://doi.org/10.1109/VLSID.2016.130

[14] M. Gautschi, M. Muehlberghuber, A. Traber, S. Stucki, M. Baer, R. Andri, L. Benini, B. Muheim, and H. Kaeslin, "SIR10us: A tightly coupled elliptic-curve cryptography co-processor for the OpenRISC," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. Zurich, Switzerland: IEEE, Jun. 2014, pp. 25–29. [Online]. Available: http://ieeexplore.ieee.org/document/6868626/

[15] M. Gautschi, P. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gurkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. PP, Aug 2016.

[16] T. R. Gross, N. P. Jouppi, J. L. Hennessy, S. Przybylski, and C. Rowen, "A Retrospective on "MIPS: A Microprocessor Architecture"," 2016.

[17] J. Hennessy, N. Jouppi, F. Baskett, and J. Gill, "MIPS: a VLSI processor architecture," in *VLSI Systems and Computations*. Springer, 1981, pp. 337–346.

[18] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill, "MIPS: A microprocessor architecture," *ACM SIGMICRO Newsletter*, vol. 13, no. 4, pp. 17–22, 1982.

[19] B. Keller, M. Cochet, B. Zimmer, J. Kwak, A. Puggelli, Y. Lee, M. Blagojevic, S. Bailey, P. Chiu, P. Dabbelt, C. Schmidt, E. Alon, K. Asanovic, and B. Nikolic, "A RISC-V processor soc with integrated power management at submicrosecond timescales in 28 nm FD-SOI," *J. Solid-State Circuits*, vol. 52, no. 7, pp. 1863–1875, 2017. [Online]. Available: https://doi.org/10.1109/JSSC.2017.2690859

[20] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, Mar. 2005. [Online]. Available: http://ieeexplore.ieee.org/document/1453485/

[21] G. Konstadinidis, M. Tremblay, S. Chaudhry, M. Rashid, P. Lai, Y. Otaguro, Y. Orginos, S. Parampalli, M. Steigerwald, S. Gundala, R. Pyapali, L. Rarick, I. Elkin, Y. Ge, and I. Parulkar, "Architecture and Physical Implementation of a Third Generation 65 nm, 16 Core, 32 Thread Chip-Multithreading SPARC Processor," *IEEE Journal of*

Solid-State Circuits, vol. 44, no. 1, pp. 7–17, Jan. 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4735553/

[22] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanovic, and K. Asanovic, "A 45nm 1.3ghz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators," in *ESSCIRC 2014 - 40th European Solid State Circuits Conference, Venice Lido, Italy, September 22-26, 2014*, 2014, pp. 199–202. [Online]. Available: https://doi.org/10.1109/ESSCIRC.2014.6942056

[23] A. S. Leon, K. W. Tam, J. L. Shin, D. Weisner, and F. Schumacher, "A Power-Efficient High-Throughput 32-Thread SPARC Processor," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 1, pp. 7–16, Jan. 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4039591/

[24] A. Lopez-Parrado and J.-C. Valderrama-Cuervo, "OpenRISC-based System-on-Chip for digital signal processing," in *2014 XIX Symposium on Image, Signal Processing and Artificial Vision*. Armenia, Colombia: IEEE, Sep. 2014, pp. 1–5. [Online]. Available: http://ieeexplore.ieee.org/document/7010123/

[25] T. Marena, "Risc-v: high performance embedded swerv™ core microarchitecture, performance and chips alliance," Western digital Corporation, Tech. Rep., Apr 2019. [Online]. Available: https://content.riscv.org/wp-content/uploads/2019/04/RISC-V_SweRV_Roadshow-.pdf

[26] N. Mehdizadeh, M. Shokrolah-Shirazi, and S. G. Miremadi, "Analyzing fault effects in the 32-bit OpenRISC 1200 microprocessor," in *2008 Third International Conference on Availability, Reliability and Security*. IEEE, Mar. 2008, pp. 648–652. [Online]. Available: http://ieeexplore.ieee.org/document/4529404/

[27] A. Menon, S. Murugan, C. Rebeiro, N. Gala, and K. Veezhinathan, "Shakti-t: A RISC-V processor with light weight security extensions," in *Proceedings of the Hardware and Architectural Support for Security and Privacy, HASP@ISCA 2017, Toronto, ON, Canada, June 25, 2017*, 2017, pp. 2:1–2:8. [Online]. Available: https://doi.org/10.1145/3092627.3092629

[28] V. Patil, A. Raveendran, P. M. Sobha, A. D. Selvakumar, and D. Vivian, "Out of order floating point coprocessor for RISC V ISA," in *19th International Symposium on VLSI Design and Test, VDAT 2015, Ahmedabad, India, June 26-29, 2015*, 2015, pp. 1–7. [Online]. Available: https://doi.org/10.1109/ISVDAT.2015.7208116

[29] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "Design and evaluation of smallfloat SIMD extensions to the RISC-V ISA," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, 2019, pp. 654–657. [Online]. Available: https://doi.org/10.23919/DATE.2019.8714897

[30] M. Tremblay and S. Chaudhry, "A Third-Generation 65nm 16-Core 32-Thread Plus 32-Scout-Thread CMT SPARC® Processor," in *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*. San Francisco, CA, USA: IEEE, Feb. 2008, pp. 82–83. [Online]. Available: http://ieeexplore.ieee.org/document/4523067/

[31] A. Waterman and K. Asanović, "The risc-v instruction set manual," The RISC-V Foundation, Tech. Rep., May 2017. [Online]. Available: https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

[32] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7ghz 64bit risc-v core in 22nm fdsoi technology," Apr 2019.

[33] B. Zimmer, Y. Lee, A. Puggelli, J. Kwak, R. Jevtic, B. Keller, S. Bailey, M. Blagojevic, P. Chiu, H. Le, P. Chen, N. Sutardja, R. Avizienis, A. Waterman, B. C. Richards, P. Flatresse, E. Alon, K. Asanovic, and B. Nikolic, "A RISC-V vector processor with simultaneous-switching switched-capacitor DC-DC converters in 28 nm FDSOI," *J. Solid-State Circuits*, vol. 51, no. 4, pp. 930–942, 2016. [Online]. Available: https://doi.org/10.1109/JSSC.2016.2519386