Microprocessors and Microsystems 61 (2018) 1–10

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

A low-cost synthesizable RISC-V dual-issue processor core leveraging the compressed Instruction Set Extension

Karyofyllis Patsidis[a], Dimitris Konstantinou[a], Chrysostomos Nicopoulos[b], Giorgos Dimitrakopoulos*,[a]

[a] Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece
[b] Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus

ARTICLE INFO

Keywords:
RISC-V
Microprocessor design
Processor architecture
Register renaming
Instruction compression

ABSTRACT

The RISC-V Instruction Set Architecture (ISA) is becoming an increasingly popular ecosystem for both hardware and software development. In this article, we investigate one of RISC-V's most versatile ISA extensions, which allows for compressed 16-bit instructions to coexist with regular 32-bit instructions. While the use of instruction compression has been touted as a means to primarily reduce code density, we present another beneficial exploitation avenue: dual issuing of compressed 16-bit instructions with minimal hardware overhead. Consequently, the proposed RISC-V processor design can substantially improve instruction throughput and reduce execution times. Additionally, the new processor employs selective register renaming to specifically target the registers used under instruction compression, thereby completely eliminating unnecessary stalls due to name dependencies. Finally, the new design utilizes a partitioned register file that capitalizes on the skewed use of registers to improve energy efficiency through clock gating. Extensive hardware analysis and cycle-accurate simulations using real applications demonstrate the effectiveness of the proposed processor architecture. Dual issuing of compressed instructions is shown to often approach the performance of a full-width two-way superscalar processor, but with much higher area and power efficiency; this is of paramount importance to severely resource-restricted emerging paradigms, such as wearable devices and Internet-of-Things (IoT) environments.

1. Introduction

The RISC-V Instruction Set Architecture (ISA) has emerged over the last few years as a completely free, open-source, widely supported, and richly documented software/hardware ecosystem [1,2] suitable for hardware implementation and software development. In addition to providing a detailed ISA specification, the RISC-V initiative includes a complete software stack – with compiler tool chains and operating system ports – that can be used for both general-purpose and embedded computing. Consequently, RISC-V has enjoyed widespread proliferation within both academic and industrial circles.

One of the salient attributes of the RISC-V ISA is its inherent extensibility, which aims to facilitate extensive specialization and customization. In addition to the base integer ISA, RISC-V supports instruction-set extensions that can leverage specialized underlying hardware, such as application-specific accelerators, Digital Signal Processors (DSP), etc. Several processor cores supporting the RISC-V ISA are already available. The targeted domains range from embedded ultra-low-power Internet-of-Things (IoT) [3] to high-performance general-purpose superscalar Out-of-Order (OoO) CPUs [4,5].

As part of its innate support for ISA extensions, RISC-V employs an encoding scheme that also supports variable-length instructions. Each instruction may comprise any number of 16-bit instruction quanta, so called "instruction parcels," and, consequently, instruction alignment may occur at 16-bit boundaries. The provided standard compressed ISA extension – called the "C extension" – provides compressed 16-bit instructions, which can markedly improve code density. In fact, the reduction in code size is mentioned in the RISC-V instruction set manual as the prime motivator for using compressed instructions. However, in this article, we aim to utilize the built-in support for compressed instructions in RISC-V for a different purpose: to improve system performance by increasing throughput. In addition to the consequent advantages of having a smaller code size, we hereby present another way to reap benefits from the presence of compressed instructions.

Specifically, we present an in-order RISC-V processor architecture that supports *dual issue of compressed 16-bit instructions* with minimal hardware overhead. Without increasing the fetch width of the processor, which is 32 bits wide, the proposed design can issue – when

* Corresponding author.
E-mail address: dimitrak@ee.duth.gr (G. Dimitrakopoulos).

https://doi.org/10.1016/j.micpro.2018.05.007
Received 4 January 2018; Received in revised form 12 April 2018; Accepted 15 May 2018
Available online 20 May 2018
0141-9331/

Table 1
The key technical attributes of various open-source RISC-V processors that are currently available, and how they compare to the proposed processor design. Entries suffixed with an asterisk (*) refer to parameters/attributes that are not explicitly reported in the provided documentation of the corresponding processor.

Core/Property:	ISA Sets	Pipeline Stages	Out Of Order	Renaming	Caches	Branch Prediction	Issue Width
Rocket [1]	RV32/64/MAFD	5	in-order	no	yes	yes	1
ORCA [6]	RV32IM	4,5	in-order	no	no	no	1*
PULPino [3]	RV32IMCF	4	in-order	no	no	no	1
OPenV/mriscv [7]	RV32I	3*	in-order	no	no	no	1
VexRiscv [8]	RV32IM	5	no*	no	yes	no*	1*
Roa Logic RV12 [9]	RV32/64IAMCE	4	in-order	no*	yes	yes	1*
SCR1 [10]	RV32I/E[MC]	2 upto 4	no*	no	no*	no*	18
BOOM [4,5]	RV64MAFD	6	ooo	yes	yes	yes	1,2,4
Shakti (Family) [11]	RVEIMCSHTN	3 upto 8+	io/ooo	y/n	y/n	y/n	various
Z-Scale [12]	RV32IM	3	in-order	no	yes	no*	1
This Work	RV32IMCF	6	in-order	selective	yes	yes	2

possible – two 16-bit compressed instructions per cycle. To the best of our knowledge, this is the first work that proposes dual-issue functionality specifically for compressed RISC-V instructions. The proposed design aims to increase performance in a very cost-effective manner, thereby targeting environments with stringent hardware area/power budgets that still demand increasingly higher performance.

By detecting the presence of 2 compressed instructions in a single 32-bit instruction chunk, the new design can improve processing throughput by simultaneously issuing both instructions to execution. Detailed profiling analysis of several benchmark applications using the evaluation framework presented in Section 4 indicates that the percentage of compressed instructions in the compiled code produced by the RISC-V compiler is very high, ranging from 37% to 76% (56%, on average). This observation serves as a primary motivation for the use of a dual-issue design specifically tailored to compressed instructions. Since (a) there is an abundance of compressed instructions in the instruction stream, and (b) two consecutive compressed instructions have the same fetch width as a single 32-bit instruction, the augmentation of dual-issue functionality for compressed instructions could constitute a very cost-effective way to substantially increase the processor's throughput. On the contrary, the alternative approach of employing a wider, superscalar design incurs significant hardware cost, which leads to reduced area/power efficiency.

Furthermore, the proposed design capitalizes on another startling observation The RISC-V ISA specification forces the use of a small group of specific registers for some classes of compressed instructions. Consequently, under instruction compression, the vast majority of the generated instructions use only a small subset of the available integer registers, namely 8 out of the 32 available architectural registers, which are available for use by compressed instructions. Specifically, when instruction compression is enabled, those 8 specific registers account for 57% to 85% (72%, on average) of all the registers used by the compiler. In other words, even a large portion of the non-compressed instructions still use the same 8 registers supported by the compressed instructions. Obviously, this unbalanced use of the pool of available registers creates an inordinate amount of unnecessary name dependencies, write-after-write and write-after-read, which hamper performance. To address this issue, the processor design proposed in this work employs selective register renaming for only those 8 "hot" registers, the use of which is spread out to a total of 16 registers, i.e., the 8 existing ones plus 8 additional ones.

The highly skewed use of registers when using instruction compression allows for another optimization. The register file can be partitioned into two segments: one housing the 8 "hot" registers plus the 8 additional physical registers used for renaming, and the other housing the remaining 24 architectural registers. In this way, the latter register partition – that is not used so frequently under instruction compression – may be appropriately clock-gated for improved energy efficiency.

The new in-order, dual-issue microprocessor supporting the RISC-V ISA was fully implemented in System Verilog, and verified using Universal Verification Methodology (UVM) test-benches. To facilitate detailed hardware evaluation, the fully functional processor was synthesized with a commercial 45 nm standard-cell library using the Cadence digital implementation flow. The hardware analysis is complemented with cycle-accurate simulations at the RTL level of several benchmark applications, which validate the efficiency of the new mechanisms. The evaluation includes comparisons to existing in-order [1] and out-of-order [4,5] state-of-the-art RISC-V-based designs. It is demonstrated that the proposed architecture can regularly approach the performance of a full-width two-way superscalar processor, albeit with much higher area/power efficiency, which is instrumental in rapidly emerging resource-constrained domains, such as wearable devices and IoT.

The rest of the paper is organized as follows: Section 2 presents related work in the domain of microprocessors supporting the RISC-V ISA. Section 3 introduces the proposed processor core and all the mechanisms that use the RISC-V compressed ISA extension to improve performance and energy efficiency. Section 4 describes the employed evaluation framework, and, subsequently, presents the evaluation results and accompanying analysis pertaining to both the performance and the hardware cost of the new processor core. Finally, concluding remarks are made in Section 5.

2. Related work

There is a rich body of literature revolving around the RISC-V ecosystem, with papers proposing different variants of processor cores, and/or architectural platforms integrating RISC-V cores with application-specific accelerators. Table 1 summarizes key technical attributes of various RISC-V processors that are currently available. Note that this is by no means an exhaustive list, but it provides an indication of the prevailing tendencies in the RISC-V domain.

The Berkeley Rocket chip SoC generator [1,13] can generate different instantiations of the Rocket RISC-V processor core. The Rocket processor has already been used within the Celebrity SoC [14], which is a parallel accelerator fabric for embedded applications that includes a neural network accelerator implemented as a Rocket custom co-processor. Similarly, VELOUR is a very-low voltage and resilient heterogeneous RISC-V-based system [15] that uses a Rocket processing core with a tightly-coupled deep neural network accelerator and another coprocessor/accelerator. The Rocket processor has also been reconstructed as a so called Labeled von Neumann Architecture [16], which is a new software-defined computer architecture concept. This Labeled-RISC-V architecture uses a labeling mechanism and programmable label-based policies to yield hardware with software-defined functionality [16].

The PULP platform [17] is a many-core accelerator aimed at ultra-low power processing applications, such as those encountered in IoT environments. The PULP platform has been used to generate processing cores implementing the RISC-V ISA [3,18,19]. Additionally, PULP is the

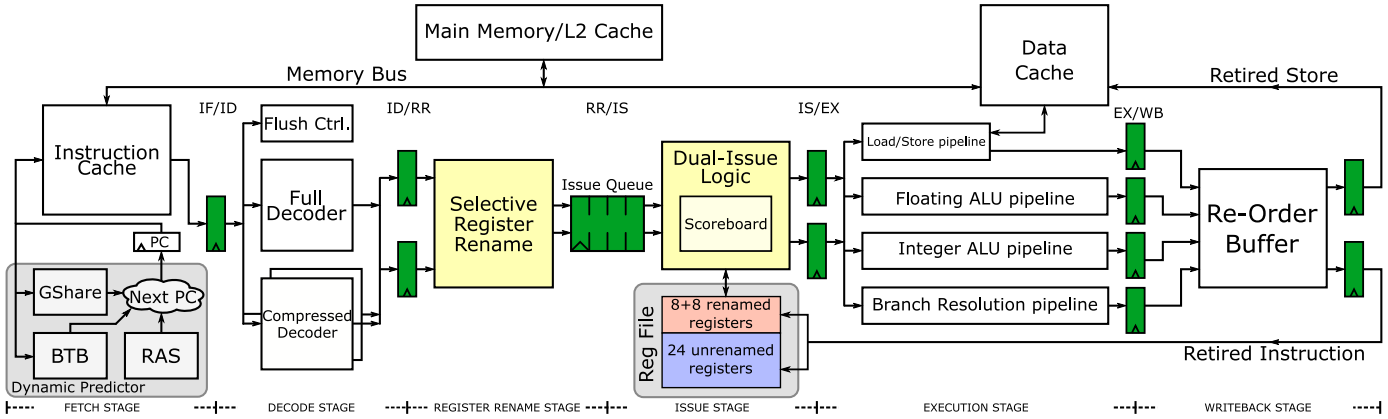

Fig. 1. A high-level overview of the proposed processor's micro-architecture. The processor is implemented in a 6-stage pipelined organization.

underlying platform used in HERO, an FPGA-based platform comprising clusters of RISC-V soft cores [20].

GRVI is a simple RISC-V scalar core that forms the key ingredient in the GRVI Phalanx [21] FPGA-based massively parallel accelerator framework. Groups of GRVI cores and specialized accelerator cores form shared-memory compute clusters. Similarly, Taiga is a RISC-V 32-bit soft processor that can be used in configurable FPGA-based multicore frameworks used for heterogeneous computing [22].

Orca is another simple RISC-V scalar core [6], intended to target FPGAs. It was built to host the Vectorblox's Proprietary Matrix processor, MXP [23], although it can also work as a standalone core. The mriscv design is a complete microcontroller featuring a RISC-V core [7]. It offers ADCs, DACs, GPIO, and SPI interfaces. The VexRiscv is a RISC-V core [8] that offers support for interactive debug and extensive statistics of performance/area/frequency obtained from synthesis for various FPGAs. The Roa Logic RV12 core [9] is a highly-configurable core, featuring a Harvard architecture for simultaneous instruction and data memory accessing. It also features an optimized folded 4-stage pipeline, for improved efficiency. The Syntacore Scr1 [10] is a core written in SystemVerilog, optimized for area and power. It also includes an integrated IRQ controller and support for advanced debugging. The Z-Scale is a tiny 32-bit RISC-V core [12] from Berkeley, suited for microcontroller and embedded systems, comprised of only 3 pipeline stages. Currently, the core is deprecated and does not support the latest ISA specifications. Yarvi [24] and LowRisc [25] are two other open-source RISC-V cores, but no public information is available besides their source code. The Riscy repository [26] provides a collection of open-sourced RISC-V processors that can be used in different simulation frameworks, or in FPGAs.

Various extensions to the RISC-V ISA have also been proposed. For example, the processor presented in [18] introduces SIMD/DSP instruction extensions to increase the processor's energy efficiency and make it suitable for near-threshold operation. The RISC-V instruction set extension described in [27] adds support for the custom programming model of MIT's Fresh Breeze system architecture.

The Simty processor [28] is a more specialized RISC-V architecture that supports the Single Instruction Multiple Threads (SIMT) execution model. Simty is a massively multi-threaded RISC-V processor core implementing the SIMT execution model purely at the microarchitecture level.

In addition to RISC-V processors targeting embedded applications, researchers have also targeted the high-performance general-purpose computing domain, building superscalar, out-of-order RISC-V processors. The two versions of the BOOM processor exemplify these efforts [4,5].

Despite the plethora of research efforts in the RISC-V domain, there is no work – to the best of our knowledge – to have examined the potential of the standard compressed ISA extension to RISC-V. In this paper, we perform an in-depth exploration of the capabilities of the RISC-V compressed ISA extension and propose very lightweight micro-architectural modifications to exploit the full-potential of compressed instructions.

The concept introduced in this work of simultaneously executing two compressed RISC-V instructions within the microprocessor's datapath is similar in vein to the VLIW mode of the non-RISC-V processor presented in [29]. The 32-bit VLIW RISC core proposed in said paper has a double datapath that can execute a pair of 16-bit instructions fetched concurrently as a 32-bit VLIW word. However, being a VLIW design, the processor in [29] relies entirely on the compiler to resolve dependencies, while the pair of 16-bit instructions can never be separated during the execution flow, which means that any stall affects both instructions at the same time. On the contrary, our approach performs scheduling of compressed instructions dynamically in hardware, and the two instructions are fully decoupled during execution and are even allowed to complete execution out-of-order.

3. The processor micro-architecture

The processor is implemented in a 6-stage pipelined organization that includes the following stages: (a) Instruction Fetch (IF), (b) Instruction Decode (ID), (c) Register Renaming (RR), (d) Instruction Issue (IS), (e) Execution (EX), and (f) Write-Back (WB). A high-level overview of the proposed processor's micro-architecture is depicted in Fig. 1. Instructions are fetched and issued for execution in program order. The write-back/retirement stage is also in-order. However, the execution can be completed out-of-order.

The duration in cycles of the EX stage is variable and it depends on the operation being executed. Table 2 summarizes the EX latencies of the various instruction types. For some instructions, the EX stage either requires several cycles to complete, or it is further pipelined in several sub-stages. Consequently, the overall pipeline depth is not 6 stages for all instructions; some instructions will experience a longer pipeline depth, due to a higher latency in their EX stage. Simple instructions, such as branches, logical operations, and additions/subtractions, take only a single cycle to complete their EX stage, i.e., they experience the nominal 6-stage pipeline. Division needs 16 cycles to complete its EX

Table 2
The execution (EX) latencies of the different instruction types.

Instruction type	Latency (cycles)
Simple arithmetic (add/sub)	1
Multiplication (pipelined)	4
Division (non-pipelined)	16
Branches	1
Load/Store	Variable

stage, during which the pipeline is busy, while multiplication is fully pipelined, with an EX latency of 4 cycles. Finally, the execution latency of memory operations is not constant, because it depends on the state of the data cache. The EX latency of memory operations can vary from a single cycle, up to several cycles, if a data cache miss occurs.

3.1. Front-end

The frond-end of the microprocessor is responsible for the fetching and decoding of the instructions. The instructions are subsequently fed to the back-end of the processor core for execution and retirement. The instruction-fetch sub-system of the front-end comprises the instruction cache and a dynamic predictor module. For optimal performance, the fetch stage should – ideally – provide a continuous stream of useful instructions, i.e., instructions on the correct execution path. To enable such seamless and continuous flow of instructions, the predictor module predicts the path that each branch will take, while the instruction cache reduces the penalty of the memory accesses. The synergistic symbiosis of these two elemental sub-systems is critical in achieving high performance. In each clock cycle, the Program Counter (PC) address is supplied in parallel to both the instruction cache and the dynamic predictor, from which the fetched instruction and the next PC address are procured, respectively.

The dynamic predictor itself utilizes three sub-modules, as illustrated in Fig. 1. The Branch Target Buffer (BTB) records the target PC address of each branch instruction, in order to expedite the determination of branch-taken addresses. Each of the BTB's entries holds a tag and a full 32-bit target address. Its indexing is performed by supplying the current PC address. If a tag match is identified, the predicted target address is the one stored in the matched entry.

The correlated branch-direction predictor maintains a history of the outcomes of previous occurrences of each branch. Based on this history, the predictor can guess the direction, i.e., taken/non-taken, of the current branch. The predictor is indexed using the current PC address. Each entry contains several potentially saturated counters. One of these counters is picked using the global branch history – recorded in a separate register – XORed with some bits of the PC address, thereby adopting a *Gshare* indexing scheme. The state of this counter will provide the final prediction. Recall that only a taken/non-taken prediction is generated by this sub-module. The actual target address of the branch is retrieved from the aforementioned BTB sub-module.

Finally, the Return Address Stack (RAS) stores the return addresses of the decoded function calls. When the function's return instruction is encountered, the entry popped from the RAS is used as the next PC address. The RAS is a complementary structure to the other two sub-modules of the predictor, and its aim is to improve the predictor's accuracy when dealing with branch instructions used for function calls/returns.

The instruction cache is a typical blocking cache structure. In the case of a hit, the access is completed within the same cycle. However, if a miss occurs, the cache "blocks" until the requested data is provided by the lower level of the memory hierarchy. The replacement policy uses a pseudo-LRU scheme to choose which block will be evicted.

The fetch width is constant at 32 bits per cycle. Fetches at the edge of cache blocks are converted into two-cycle operations, in which the data is retrieved in packets and then re-assembled. The instruction alignment is done at 16-bit boundaries. This implies that, within the 32 fetched bits, one may find one full 32-bit instruction, or two compressed 16-bit instructions, or even invalid parts of instructions, if misalignment occurs. Any misalignment is detected and signalled by the decoder, so that the fetch sub-system can redirect and re-align to the correct address. Redirection also occurs if a branch resolution indicates a misprediction.

The Instruction Decode (ID) stage is responsible for decoding the fetched 32-bit instruction chunk. It contains one decoder for full-length 32-bit instructions, and two decoders for compressed 16-bit instructions. In RISC-V ISA the fields are always encoded in the same place inside the instruction body, which makes the decoding fairly straight-forward. As previously mentioned, the ID stage is also responsible for detecting misalignment and invalid instructions, which generate appropriate signalling for the IF stage to realign and resume.

It should be noted that adding support for compressed instructions is not a straight-forward task, as also described in [30]. Not all 16-bit instructions map fully to the corresponding 32-bit ones. This means that the pipeline must be aware of the nature of the instructions, i.e., whether they were originally compressed. The same applies to the program flow controllers and predictors, which should also track whether any given instruction was originally compressed or not. We tackle this problem by efficiently allocating internal micro-op codes to the instructions, and by tracking the program flow in the ID stage, where all necessary information is readily available. This removes the need for any extra book-keeping in the pipeline.

3.2. Register Renaming and dispatch to execution

Upon completion of the ID stage, the decoded instructions are fed into the Register Renaming (RR) stage. The inclusion of such a subsystem was deemed necessary after detailed profiling analysis of several benchmark applications, which revealed a key attribute pertaining to how the available registers are utilized by the RISC-V compiler. The details of our evaluation framework and the profiling analysis of the benchmarks are presented in Section 4. It turns out that, when using instruction compression, the RISC-V ISA specification dictates the use of predominantly 8 specific registers for some classes of compressed instructions, namely registers x8–x15. These 8 registers are used, on average, 72% of the time across all examined benchmarks, and, naturally, they give rise to a huge amount of name hazards/dependencies, write-after-write and write-after-read.

To mitigate this artificial bottleneck and reap all the potential benefits of using compressed instructions, the proposed processor design employs selective register renaming targeting only registers x8–x15. Through register renaming, the 8 heavily used architectural registers are spread out to 16 physical registers. An overview of the proposed selective register renaming scheme is shown in Fig. 2. Selective – rather than full – renaming allows for significant area savings,

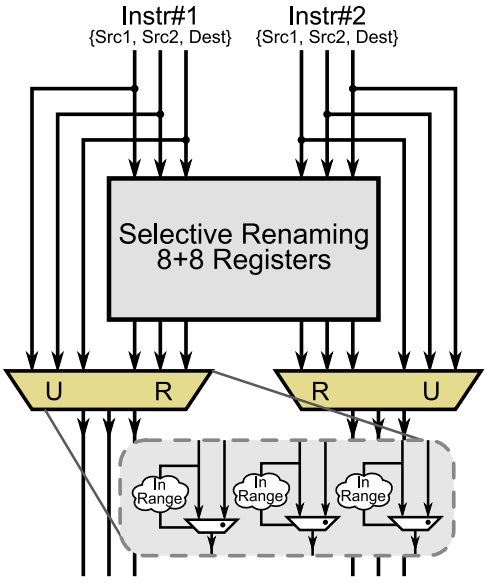

Fig. 2. The proposed selective register renaming scheme. The 8 heavily used architectural registers (x8–x15 in the ISA) are spread out to 16 physical registers through renaming ('R' paths). The other architectural registers are not renamed ('U' paths, i.e., un-renamed). Note that up to two instructions can be renamed per cycle.

since only 40 physical registers are needed in total, 32 existing + 8 extra for renaming, instead of the 64 that would be required if all 32 architectural registers were to be renamed. Furthermore, due to the limited scope of the renaming, all the structures required for renaming are significantly smaller.

Up to two instructions can be renamed per cycle, to enable dual-issue functionality later on, as shown in Fig. 2. The renaming system utilizes a Register Alias Table (RAT) that keeps track of the mapping of the 8 "hot" architectural registers to the 16 physical ones. Since the RAT is corrupted on mis-predicted branches, the RAT uses a checkpoint-based repair mechanism. Each new renamed branch triggers a new checkpoint capture. If the branch ends up being mis-predicted, the RAT can be repaired within a single cycle, by restoring the corresponding checkpoint. Up to four checkpoints are supported, and, therefore, up to four branches can be in-flight simultaneously. A free-list structure is also used, which is a simple queue holding the identifiers of the free physical registers that can be allocated to the incoming instructions.

The renamed instructions then pass through the Issue Stage (IS), before being dispatched for execution. The IS stage is built around a simple score-boarding mechanism, which keeps track of the status of each physical register. The score-board has a total of 40 entries, one for each physical register. It keeps track of each register's usage, as well as the location of the latest data. If some pertinent conditions are satisfied, up to two instructions may be dispatched in-order for execution. The conditions that must be met before dispatching are: (a) the source data must be available, (b) the functional units must be ready to accommodate new instructions, and (c) no conflicts – dependent data, or use of the same functional unit – must exist between the instructions. The supported dual-issue functionality allows for the simultaneous issuing of two compressed 16-bit instructions without the need to double the fetch width. Thus, in addition to exploiting compressed instructions to reduce the code size, the proposed processor design also reaps performance – throughput – benefits through the dual issuing of two consecutive compressed instructions to execution.

Instructions that are dispatched to execution obtain their source operands from the Register File, as indicated in Fig. 1. In the proposed implementation, the register file design capitalizes on the skewed use of the registers when instruction compression is employed. Recall that the registers used most of the time are 16 specific registers, i.e., registers x8–x15 and the 8 physical registers used in the selective register renaming scheme. The remaining 24 architectural registers are used markedly less often. This operational characteristic has motivated us to design a partitioned register file, which is split in two segments: one housing the 16 frequently used registers, i.e., the selectively renamed registers, and the other housing the 24 registers that are not accessed so frequently, i.e., not renamed. The architecture of the employed partitioned register file is abstractly illustrated in Fig. 3. Essentially, the register file is partitioned into a "hot" segment that is primarily used during execution, and a "cold" segment that is used more sporadically.

3.3. Back-end

The back-end of the processor core consists of the Execution (EX) and the Write-Back (WB) stages. There are four functional units in the EX stage, each tasked with different operations. The first unit is the integer ALU. All its operations are pipelined and, therefore, it can accept a new instruction every cycle. The only operation that will cause the unit to block is division, which is not pipelined. The second unit is the branch-resolve unit, responsible for the resolution of all branch instructions. The execution of this particular unit always finishes within one cycle. The third unit is the load/store unit that has a direct connection to the data cache. Since RISC-V is a load-store ISA, the data cache can also provide lock-free operation. This means that upon misses, it can continue to serve new instructions while waiting for the data to arrive from the next level of the memory hierarchy. This was a deliberate design choice, so that the performance of the processor could be somewhat decoupled from the performance of the memory system. To achieve non-blocking operation, we employ buffers (load/store/wait) that store relevant information of the instructions that missed, until they can be served. The fourth functional unit is the floating-point ALU, which, in the current implementation, is missing. The whole pipeline organization is fully ready to support floating-point instructions; the only remaining module to be implemented in the next incarnation of the processor is the floating-point unit.

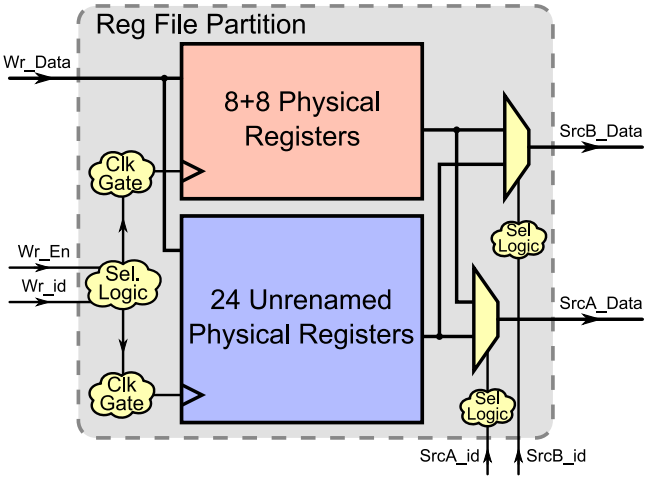

Fig. 3. The proposed partitioned register file comprising a "hot" segment – top partition – that is primarily used during execution for the selectively renamed registers, and a "cold" segment – bottom partition – that is used more sporadically for the registers that are not renamed.

The WB stage is facilitated through the use of a Re-Order Buffer (ROB). This pipeline stage uses the well-established "data-in-ROB" scheme, whereby all execution results are temporarily stored within the ROB, until the instructions can commit/retire. The ROB maintains the correct program order during the retirement of the instructions, and allows for speculative execution beyond branches. In each clock cycle, only the oldest – in terms of program order – instruction can commit and modify the architectural state of the processor. Retirement includes writing the results to the physical register file, or saving the data in the data cache, in the case of store instructions. Because the retirement rate is limited to one instruction per cycle in the current implementation, the effective maximum throughput of the pipeline is also set to one instruction per cycle.

4. Evaluation results

4.1. Evaluation framework

In this section, we perform a detailed evaluation of the proposed RISC-V microprocessor core, and we compare it – in terms of performance and hardware cost – with two existing, state-of-the-art RISC-V-based cores: (a) the Rocket core [1], which is a single-issue, in-order processor with no compressed instructions, no register renaming, and no register-file partitioning; and (b) the BOOM core [4,5], which is a two-way superscalar processor with out-of-order execution and register renaming, but no support for compressed instructions. In all comparisons, the Rocket core [1] serves as the baseline low-cost processor, while – at the other end of the spectrum – the BOOM core [4,5] serves as a reference high-performance processor with high hardware cost.

The proposed core design was described in SystemVerilog. It was debugged via extensive RTL simulation and with the use of UVM testbenches, both at the sub-module level and at the system level. In the initial phase of validation, random instruction sequences were employed to stress each component and identify obscure corner-case bugs.

Subsequently, several real benchmark applications were compiled and executed to completion at the RTL level to prove the core's correct execution flow. The RTL simulation results were compared with the high-level RISC-V functional simulator *Spike* for correctness.

The performance of the new processor was evaluated using a suite of 11 benchmark applications, as follows: *2D-median filter, FIR filter, bubblesort, insertion sort, binary search, autocorrelation, vector-vector addition, software multiplication, dot product, factorial*, and *XOR cipher*. The selected benchmarks include a representative mix of algorithm types encountered in both the embedded and general-purpose computing domains. Three are for signal processing/filtering, four are for mathematical computations, three for data processing (search/sort), and one for data encryption.

All benchmarks were cycle-accurately executed from start to finish at the RTL level, and various salient statistics were retrieved from performance counters present in the processor. These include the standard instruction/cycle/timing counters, and more specialized trackers for hazards and stalls in the various pipeline stages, which were added to aid our detailed performance profiling analysis. All statistics were gathered after the complete execution of each benchmark. Furthermore, the computational output of each benchmark was compared between all investigated processor implementations to ensure the same execution flow. The validation comparison included the final contents of the register file and memory, and the total numbers of executed and committed instructions.

In addition to being evaluated in terms of performance, the proposed processor was also assessed in terms of its hardware cost, in order to quantify the area/power/timing overhead of the employed techniques. The hardware cost evaluation was conducted after synthesizing the RTL implementation of the core using a commercial 45 nm standard-cell library under worst-case conditions (0.8 V, 125°C), using the Cadence digital implementation flow.

4.2. Performance evaluation

As part of the performance evaluation of the proposed RISC-V processor core, the 11 previously described benchmark applications were executed to completion and various statistics were collected. Four different processor designs were evaluated: (1) the Rocket core [1] as a baseline single-issue, in-order setup, (2) a version with compressed instruction support and dual-issue functionality (called "dual") to increase the throughput in the presence of 16-bit compressed instructions, (3) a version of the core with compressed instruction support, dual-issue functionality and selective register renaming (called "dual+RR"), as described in Section 3, and (4) the full two-way superscalar out-of-order BOOM core [4,5]. The processor parameters used in our simulations are summarized in Table 3. Note that we evaluate two different cache/memory setups: (a) a memory hierarchy with only a Level-1 (L1) cache, and (b) a memory hierarchy with two levels of cache, L1 and L2. The former setup is intended to represent a low-cost lightweight system, while the latter setup represents a higher-performance system.

The obtained results are reported in Table 4. The first notable observation in Table 4 is the very high percentage of compressed instructions found in all benchmarks. When the compiler utilizes the standard compressed RISC-V ISA extension, the amount of generated compressed instructions in the resulting binary ranges from 37% to 76%, without any user intervention. It is precisely this extensive use of instruction compression that motivated the proposed dual-issue functionality: since pairs of 16-bit instructions are very often encountered in the 32-bit instruction-fetch chunks, it would be both cost-effective and advantageous to employ dual-issue functionality to increase throughput.

Indeed, the dual-issue functionality is extensively utilized in the vast majority of the applications, as indicated by the results in the fourth column of Table 4. On average, 28% and 27% of all issue operations are dual in the processors supporting dual-issue and dual-issue+renaming, respectively.

Table 3
The simulated processor/system parameters.

Parameter	Value
General parameters	
Instruction-Q Size	4 entries
ROB Size	8 entries
Max Branches in-flight	4 (applicable only with RR)
Memory parameters	
IC Lines	256
IC Line Size	256
IC Associativity	2-way
DC Lines	256
DC Line Size	256
DC Associativity	4-way
L2 Lines	2048
L2 Line Size	512
L2 Latency	10 cycles
L2 Associativity	1-way
Main Memory Lines	2048
Main Memory Line Size	512
Main Memory Latency	50 cycles
Predictor parameters	
BTB Size	128 entries
Gshare Size	128 entries
Branch Global History	2 bits

A most surprising observation, however, is made when looking at the usage of the eight architectural registers x8–x15, as a percentage of all register usage by the compiler (third column in Table 4). These 8 registers are available for use by compressed instructions. In general, under instruction compression, the RISC-V ISA specification imposes the use of primarily those 8 registers for some classes of compressed instructions. In fact, the usage of said registers ranges from 57% to 85% across the 11 examined benchmarks. We surmise that the reason for this highly skewed use is the fact that those registers are preferred even by the non-compressed instructions, to perhaps minimize data movement between compressed and uncompressed instructions.

As a direct consequence of this heavily unbalanced use of the register resources, an exceedingly high number of name hazards – write-after-write and write-after-read – are observed during execution. On average, applications are stalled 17% of the time due to name hazards as reported in the column '% Stalls due to Hazards' of Table 4. It is interesting to note that the percentage of stalls due to hazards increases slightly under dual-issue functionality, as compared to the baseline single-issue case. This is because the increased throughput provided by dual-issuing further accentuates the bottleneck created by the name hazards. To tackle this bottleneck, we employ selective register renaming targeting specifically the 8 "hot" registers x8–15. By spreading out the usage of those 8 architectural registers to a total of 16 physical registers, we completely eliminate the stalls due to hazards, as shown in Table 4 for the processor supporting dual-issuing and register renaming.

We also quantified the percentage of time the core is stalled due to an empty instruction queue. The latter becomes empty when the instruction fetching mechanism cannot keep up with the back-end of the CPU. The two right-most columns of Table 4 report this percentage for both the L1-only system, and the system with L1 and L2 caches. In the second case, the L2 cache is assumed to have a 10-cycle access latency.

As expected, the addition of dual-issuing and register renaming to the system with only one level of cache leads to an increase in the stalls due to an empty instruction queue, since the back-end is now much more effective in consuming, i.e., executing incoming instructions. Consequently, the front-end – instruction fetching – now impedes the execution flow. In the case of the processor with dual-issuing and register renaming, the average percentage of stalls is 32%. This particular analysis pinpoints the fetch bandwidth as an excellent target in the pursuit of further performance improvements. One way to mitigate the limitation of the narrow fetch bandwidth without increasing the fetch

Table 4
The performance results for all benchmark applications when executed on the four different investigated processor configurations. All results are derived from a system with only one level of cache, except those in the right-most column.

Benchmark & Processor Config.	% Instruction Compr.	% Dual-Issue Utilization	% Stalls due to Hazards	% Stalls due to Empty-Q	% Stalls Empty-Q With L2 Cache
2D median -Rocket [1]	0%	0%	54%	0%	0%
2D median -dual	47%	20%	61%	0%	0%
2D median -dual + RR	47%	14%	0%	41%	16%
2D median -Boom [4,5]	0%	14%	0%	6%	2%
multiply -Rocket [1]	0%	0%	4%	12%	12%
multiply -dual	48%	40%	2%	43%	44%
multiply -dual + RR	48%	40%	0%	51%	51%
multiply -Boom [4,5]	0%	46%	0%	22%	20%
bubblesort -Rocket [1]	0%	0%	1%	29%	20%
bubblesort -dual	76%	46%	1%	33%	27%
bubblesort -dual + RR	76%	47%	0%	36%	31%
bubblesort -Boom [4,5]	0%	52%	0%	37%	27%
binary search -Rocket [1]	0%	0%	15%	29%	21%
binary search -dual	37%	10%	17%	36%	22%
binary search -dual + RR	37%	8%	0%	46%	38%
binary search -Boom [4,5]	0%	13%	0%	41%	38%
insertion sort -Rocket [1]	0%	0%	9%	6%	6%
insertion sort -dual	54%	20%	5%	18%	17%
insertion sort -dual + RR	54%	20%	0%	8%	8%
insertion sort -Boom [4,5]	0%	20%	0%	7%	9%
autocorrelation -Rocket [1]	0%	0%	20%	6%	2%
autocorrelation -dual	51%	55%	30%	6%	3%
autocorrelation -dual + RR	51%	55%	0%	8%	5%
autocorrelation -Boom [4,5]	0%	46%	0%	13%	5%
vvadd -Rocket [1]	0%	0%	8%	3%	2%
vvadd -dual	65%	33%	13%	3%	2%
vvadd -dual + RR	65%	33%	0%	8%	12%
vvadd -Boom [4,5]	0%	33%	0%	4%	3%
dot_product -Rocket [1]	0%	0%	12%	6%	3%
dot_product -dual	60%	39%	18%	6%	3%
dot_product -dual + RR	60%	39%	0%	6%	4%
dot_product -Boom [4,5]	0%	40%	0%	2%	2%
factorial(!100) -Rocket [1]	0%	0%	43%	13%	5%
factorial(!100) -dual	57%	1%	47%	16%	7%
factorial(!100) -dual + RR	57%	1%	0%	43%	31%
factorial(!100) -Boom [4,5]	0%	1%	0%	43%	19%
xor_cipher -Rocket [1]	0%	0%	6%	83%	51%
xor_cipher -dual	62%	23%	6%	80%	50%
xor_cipher -dual + RR	62%	19%	0%	86%	61%
xor_cipher -Boom [4,5]	0%	47%	0%	90%	70%
fir[100samples/5taps] -Rocket [1]	0%	0%	21%	14%	6%
fir[100samples/5taps] -dual	59%	18%	25%	13%	8%
fir[100samples/5taps] -dual + RR	59%	6%	0%	17%	13%
fir[100samples/5taps] -Boom [4,5]	0%	33%	0%	21%	10%
Overall Average -Rocket [1]	0%	0%	17%	18%	12%
Overall Average -dual	56%	28%	20%	23%	17%
Overall Average -dual + RR	56%	27%	0%	32%	25%
Overall Average -Boom [4,5]	0%	32%	0%	26%	19%

width is the inclusion of a second level of cache (L2) in the memory hierarchy. As indicated in the right-most column of Table 4, the percentage of time the core is stalled due to an empty instruction queue when using a system with both L1 and L2 caches drops significantly in most benchmark applications. Specifically, the average percentage of stalls drops to around 25%. The decrease in stalls corroborates the improvement in the fetching efficiency of the front-end of the processor, due to the presence of the L2 cache. This enhancement translates into corresponding Instructions Per Cycle (IPC) gains, as will be demonstrated next.

Fig. 4 reports the IPC gains achieved by the proposed processor designs using the compressed instruction set (both the version with only dual-issuing, and the one with both dual-issuing and selective register renaming), and the Rocket [1] and BOOM [4,5] cores. Fig. 4(a) refers to the system with only one level of cache, while Fig. 4(b) refers to the system with L1 and L2 caches. In both cases, the IPC gains are normalized to the baseline processor, i.e., the Rocket core [1] with no compressed instructions and no register renaming, as indicated by the black horizontal line at the Normalized IPC value of 1. As suggested by the previously described abundance of stalls due to name hazards (see Table 4), the addition of compressed instruction support and dual-issue functionality yields moderate IPC improvements in most applications, since performance is dictated and limited by the hazard stalls. Indeed, as illustrated in both Figs. 4(a) and (b), the processor with only dual-issuing capability exhibits moderate IPC gains of 7%. However, in the case of the `multiply` benchmark, the overall performance is degraded by the presence of compressed instructions. Consequently, the achieved performance – for said benchmark – is slightly worse than under the baseline Rocket processor [1]. This behavior is attributed to excessive fetch redirections triggered by misaligned fetches of compressed instructions. Such degeneracy results in a smaller instruction-fetch rate when executing this benchmark, and, therefore, worse front-end performance.

However, when dual-issuing is accompanied by selective register renaming, IPC improves substantially. Specifically, in the case of the L1-only system (Fig. 4(a)), the IPC increases by as much as 92%, with the average gain being 30% across all 11 benchmarks. The only benchmark application that exhibits a degradation in performance – of around 47%

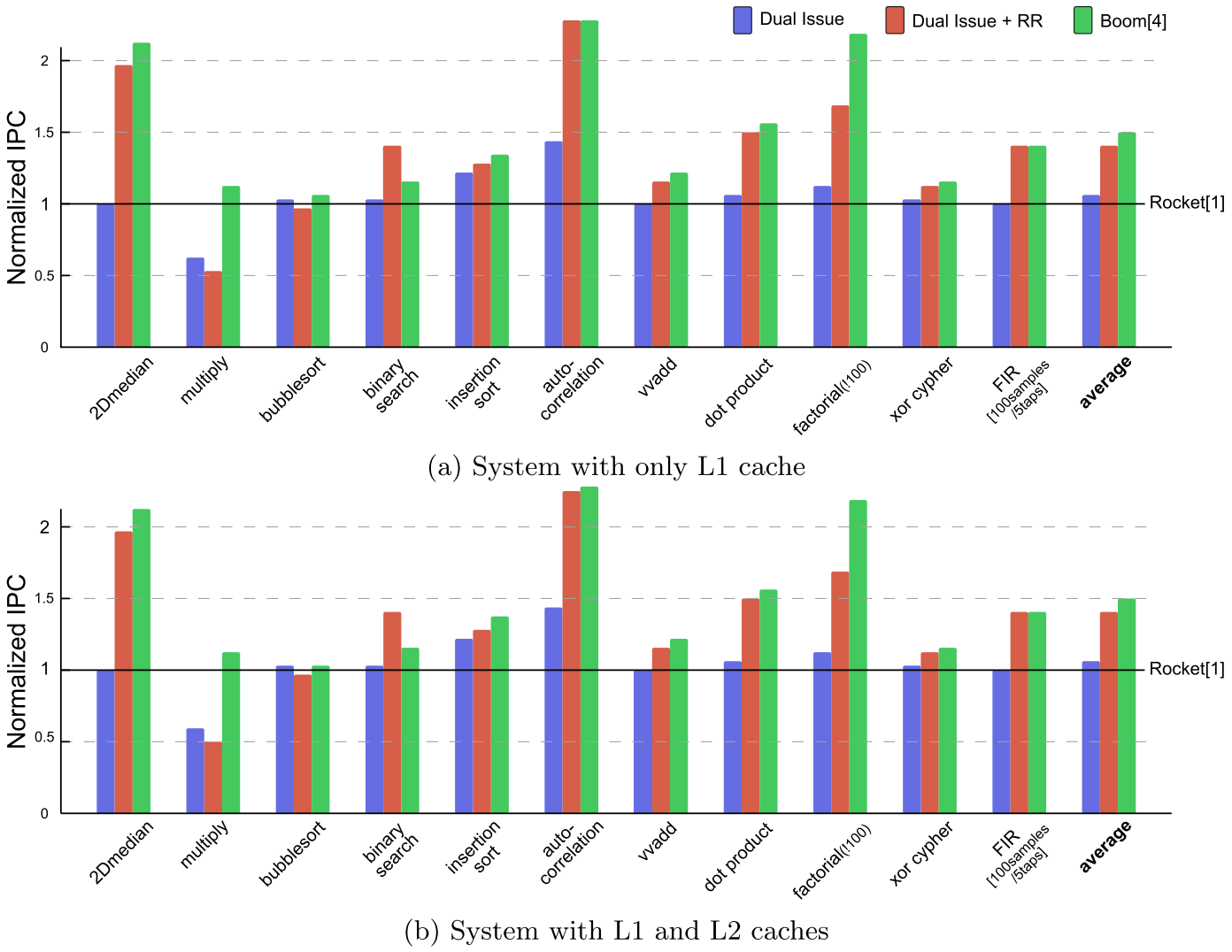

Fig. 4. The IPC gains achieved by the investigated processor designs in (a) a system with only one level of cache, and (b) a system with L1 and L2 caches. The results are normalized to the baseline single-issue Rocket [1] processor, as indicated by the black horizontal line at the Normalized IPC value of 1.

– under dual-issuing and selective register renaming is `multiply`. This anomaly is attributed to the combination of high fetch redirections, i.e., many branches and function calls, and a low number of renaming hazards. Since no hazards existed in the first place, the addition of selective register renaming cannot offer any improvement. Yet, renaming inserts an extra cycle of latency in the pipeline. This extra latency, combined with the increased fetch redirections of the program, results in a further performance degradation.

As expected, after adding a second-level cache to the system, the obtained IPC gains are even higher, as depicted in Fig. 4(b). The IPC increases by as much as 130%, with the average gain being 39% across all benchmarks.

As anticipated, the two-way superscalar BOOM architecture [4,5] provides a clear lead in performance, as compared to the other designs under investigation. Specifically, the BOOM processor can improve the IPC by as much as 129%, as compared to the baseline, while the average IPC increase across all benchmarks is 51%, when paired with an L2 memory. The very high reaped gains in some benchmark applications are the result of the clear dependency of the overall CPU performance to the performance of the front end, which tends to be the pipeline's bottleneck.

Nevertheless, the performance of the proposed design with dual issuing and selective register renaming is quite similar to BOOM's performance under most benchmarks. This is because the presence of compressed instructions increases the effective instruction fetch rate; since compressed instructions are simultaneously fetched in pairs, the front end is markedly sped up. In the presence of compressed instructions, the effective fetch rate of the proposed design is essentially the same as that in a two-way superscalar, but with half the fetch width.

It should be noted that a somewhat counter-intuitive behavior is observed under the `binary search` and `xor cipher` benchmarks. In these two cases, the proposed architecture actually manages to slightly outperform the BOOM superscalar core in some configurations. The reason is that the superscalar's fetch rate can occasionally be severely limited – even close to one 32-bit instruction per cycle – in the presence of some branch sequences that trigger multiple fetch redirections. On the other hand, the presence of compressed instructions in the proposed design implies that a 32-bit fetch often includes two instructions, thereby yielding a slightly higher overall instruction fetch rate.

4.3. Hardware cost analysis

Our goal is to evaluate the hardware complexity of a dual-issue RISC-V core with selective register renaming and a partitioned register file, versus the complexity of a single-issue baseline RISC-V processor (Rocket [1]), and that of a two-way superscalar RISC-V processor (BOOM [4,5]). Note that, natively, BOOM [4,5] is a 64-bit RISC-V processor. However, for fair comparison, in this work we investigate a

Table 5
Hardware implementation results of the four investigated processor cores (excluding their caches) at 45 nm / 0.8 V.

Design 900MHz	Area (mm^2)	Avg. Power (mW)
Rocket [1]	0.16	7.7
Dual	0.17	8.7
Dual + RR	0.18	10.7
Boom [4,5] (32-bit)	0.24	14.7

32-bit variant of the BOOM [4,5] core. The implementation results were obtained for all designs, after constraining the logic-synthesis process using the same parameters for all designs. During synthesis, we followed a mixed-Vt approach, where a mix of regular Vt, Low-Vt, and High-Vt cells were used for the various implementations. Low-Vt cells are used on the critical paths, while the rest cells were employed on the non-critical paths to reduce leakage power [31]. Power was measured after performing timing-accurate simulations of the gate-level netlists of the processors executing the benchmarks in order to prepare the switching activity information.

The maximum speed that could be obtained by the dual-issue core with selective register renaming is 900 MHz. The single-issue core can be slightly faster due to relaxed critical paths. However, to have a fair comparison, the baseline core was also optimised for the same speed. It should be noted that the delay numbers reported here correspond to a low voltage of 0.8 V, which significantly increases the delay of the circuits. Other published results for RISC-V processor cores of similar complexity [32] reveal that their frequency is also below 1 GHz when operating at 0.8 V.

The derived area/power results are summarized in Table 5. Support for instruction compression and dual-issue functionality add 6% of additional area, due to (a) the extra decoders and decoding logic needed for the compressed instructions, (b) the extra read ports needed in the register file, the scoreboard, and the instruction queue, (c) the more complex issue logic to facilitate the dual-issue functionality, and (d) the additional pipeline registers needed between the IS and EX stages. Dual-issue functionality also increases the average power consumption by 13% relative to a single-issue core.

Dual-issue functionality cannot show its full promise without the introduced selective register renaming scheme. After the addition of the selective register renaming pipeline stage, the overall area increases by 13%, as compared to the baseline processor architecture. This is due to the extra renaming structures used in the RR stage, i.e., the renaming table and free list, the extra 8 physical registers in the register file, and the new pipeline register. However, the performance gains by renaming these 8 architectural registers are quite significant. Without the use of an L2 Cache, the average IPC is increased by as much as 30%, whereas, when including an L2 cache, the average IPC increases by 39%. The additional hardware structures needed by selective register renaming scheme increase the average power consumption of the core. This increase is justified by the performance gains earned. However, the total overhead of the selective register renaming scheme is just marginal in the end, if the power contribution of the caches is taken into account. The clock-gating structure employed in the partitioned register file offers a 4% power benefit relative to a uniformly clock-gated register file. Additional benefits – in terms of leakage power – would have been observed if we could have power-gated separately the two register-file partitions (a feature not supported by our standard-cell library).

More importantly, the proposed architecture with dual issuing of compressed instructions and selective register renaming consumes less area and power than the BOOM [4,5] two-way superscalar processor. As shown in Table 5, BOOM [4,5] consumes 33% more area and 37% more power than the proposed design.

Resource-constrained compute environments demand increasingly higher performance, but in a very cost-effective manner. This is precisely the ultimate aim of the proposed design: reap as much performance as possible, with as little hardware overhead as possible. Hence, the most appropriate figures of merit in these circumstances are the area and power *efficiencies*, i.e., the IPC/mm^2 and IPC/Watt metrics, as illustrated in Fig. 5. The results are normalized to the single-issue in-order Rocket core [1], as indicated by the black horizontal line at the Normalized Area/Power Efficiency value of 1. Clearly, the proposed architecture supporting dual issuing of compressed instructions and selective register renaming ('Dual Issue + RR') achieves the highest area and power efficiencies. While the superscalar BOOM core [4,5] achieves the best overall raw performance (see Section 4.2), its substantial hardware overhead results in lower area/power efficiency. On the contrary, the proposed approach closely approaches the superscalar's performance at a significantly lower hardware cost.

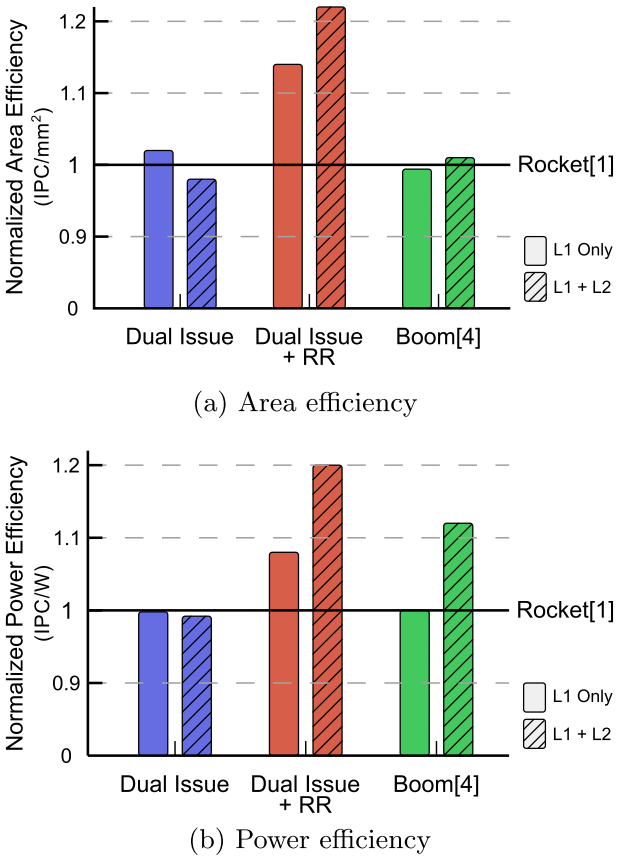

Fig. 5. The area and power efficiencies achieved by the investigated designs in systems with one and two levels of caches. The results are normalized to the single-issue in-order Rocket core [1], as indicated by the black horizontal line at the Normalized Area/Power Efficiency value of 1.

5. Conclusion

The compressed instruction set extension of the RISC-V ISA is highly versatile. In addition to the obvious benefit of code density reduction, instruction compression can also be exploited to improve throughput/performance with low hardware cost. The processor architecture introduced in this article relies on a triptych of synergistic techniques that exploit instruction compression to markedly improve system performance. The dual issuing of compressed 16-bit instructions was shown to be very cost-effective, mainly due to the fact that the fetch width remains constant. Complementing the dual-issue functionality, selective register renaming has a highly targeted and limited scope, which also results in minimal hardware overhead. Finally, the use of compressed instructions was shown to create a very skewed use of the register pool. Leveraging this attribute, the new processor design employs a

partitioned register file, which increases energy efficiency through clock gating. Together, the three aforementioned techniques yield significant improvements in performance with minimal hardware cost, as demonstrated by our extensive evaluation. Comparisons against existing state-of-the-art RISC-V designs indicate that the proposed architecture can approach the performance gains of a two-way superscalar processor at a much lower hardware cost. Consequently, the new design achieves higher area/power efficiency, which is critical in environments with scarce resources, such as those involving wearable and IoT devices.

References

[1] 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, A. Waterman, The Rocket Chip Generator, Technical Report, EECS Department, University of California, Berkeley, 2016.

[2] RISC-V Foundation, (http://www.riscv.org, Accessed: 19-12-2017).

[3] A. Traber, PULPino: A small single-core RISC-V SoC, 2016, (https://www.riscv.org/wp-content/uploads/2016/01/Wed1315-PULP-riscv3_noanim.pdf).

[4] C. Celio, D.A. Patterson, K. Asanović, The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor, Technical Report, EECS Department, University of California, Berkeley, 2015.

[5] C. Celio, P.-F. Chiu, B. Nikolic, D.A. Patterson, K. Asanović, BOOMv2: an open-source out-of-order RISC-V core, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[6] VectorBlox Computing Inc. ORCA Core, https://www.github.com/vectorblox/orca, Accessed: 19-12-2017.

[7] ONCHIP GROUP. Mrisc Core, (https://www.github.com/onchipuis/mriscv). Accessed: 19-12-2017.

[8] VexRiscv Core, (https://www.github.com/SpinalHDL/VexRiscv). Accessed: 19-12-2017.

[9] ROA Logic. RV12 Core, (https://wwww.github.com/roalogic/RV12). Accessed: 19-12-2017.

[10] Syntacore's SCR1, (https://wwww.github.com/syntacore/scr1). Accessed: 19-12-2017.

[11] Shakti Processor project. IIT Madras, (http://www.rise.cse.iitm.ac.in/shakti.html). Accessed: 19-12-2017.

[12] Z-Scale, (https://www.github.com/ucb-bar/zscale). Accessed: 19-12-2017.

[13] Freechips Project, (https://www.github.com/freechipsproject/rocket-chip). Accessed: 19-12-2017.

[14] T. Ajayi, K. Al-Hawaj, A. Amarnath, S. Dai, S. Davidson, P. Gao, G. Liu, A. Lotfi, J. Puscar, A. Rao, A. Rovinski, L. Salem, N. Sun, C. Torng, L. Vega, B. Veluri, X. Wang, S. Xie, C. Zhao, R. Zhao, C. Batten, R.G. Dreslinski, I. Galton, R.K. Gupta, P.P. Mercier, M. Srivastava, M.B. Taylor, Z. Zhang, Experiences using the risc-v ecosystem to design an accelerator-centric soc in tsmc 16nm, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[15] S. Eldridge, K. Swaminathan, N. Chandramoorthy, A. Buyuktosunoglu, A. Roelke, X. Guo, V. Verma, R.S. Joshi, M.R. Stan, P. Bose, A Low Voltage RISC-V Heterogeneous System, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[16] Z. Yu, B. Huang, J. Ma, N. Sun, Y. Bao, Labeled RISC-V: a new perspective on software-defined architecture, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[17] D. Rossi, I. Loi, F. Conti, G. Tagliavini, A. Pullini, A. Marongiu, Energy efficient parallel computing on the PULP platform with support for OpenMP, 2014 IEEE 28th Convention of Electrical Electronics Engineers in Israel (IEEEI), (2014), pp. 1–5.

[18] M. Gautschi, P.D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F.K. Gürkaynak, L. Benini, A near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices, IEEE Trans. Very Large Scale Integr. VLSI Syst. 25 (2017) 2700–2713.

[19] Pulp Platform Project. 2017. Pulpino, (https://github.com/pulp-platform/pulpino). Accessed: 19-12-2017.

[20] A.K.P. Vogel, A. Capotondi, A.M.L. Benini, HERO: heterogeneous embedded research platform for exploring RISC-V manycore accelerators on FPGA, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[21] J. Gray, A.K.P. Vogel, A. Capotondi, A.M.L. Benini, GRVI Phalanx: a massively parallel RISC-V FPGA accelerator framework, and a 1680-core, 26 MB SRAM parallel processor overlay on Xilinx ultraScale+ VU9P, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[22] E. Matthews, L. Shannon, Taiga: a configurable RISC-V soft-processor framework for heterogeneous computing systems research, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[23] VectorBlox Computing Inc. MXP™ Matrix Processor, (https://www.github.com/VectorBlox/mxp). Accessed: 19-12-2017.

[24] YARVI, (https://www.github.com/tommythorn/yarvi). Accessed: 19-12-2017.

[25] lowRISC, (https://www.github.com/lowRISC/lowrisc-chip). Accessed: 19-12-2017.

[26] MIT CSAIL's computation structures group. 2017. Riscy processors -open- sourced RISC-V processors, (https://www.github.com/csail-csg/riscy). Accessed: 19-12-2017.

[27] J.B. Dennis, W. Lim, A RISC V extension for the fresh breeze architecture, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[28] S. Collange, Simty: generalized SIMT execution on RISC-V, First Workshop on Computer Architecture Research with RISC-V (CARRV 2017), (2017).

[29] F. Campi, R. Canegallo, R. Guerrieri, Ip-reusable 32-bit vliw risc core, Proceedings of the 27th European Solid-State Circuits Conference, (2001), pp. 445–448.

[30] C. Celio, Documentation for the BOOM processor, (https://www.ccelio.github.io/riscv-boom-doc/). Accessed: 19-12-2017.

[31] T. Karnik, Y. Ye, J. Tschanz, L. Wei, S. Burns, V. Govindarajulu, V. De, S. Borkar, Total power optimization by simultaneous dual-vt allocation and device sizing in high performance microprocessors, Proc, of the 39th Annual Design Automation Conference, (2002), pp. 486–491.

[32] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, V. Stojanović, K. Asanović, A 45nm 1.3ghz 16.7 double-precision gflops/w risc-v processor with vector accelerators, 40th European Solid State Circuits Conference (ESSCIRC), (2014), pp. 199–202.

Karyofyllis Patsidis received the Diploma in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2017, where he is currently pursuing his M.Sc. degree. His research interests include computer architecture, real-time/low-power processor architectures, hardware acceleration of Neural Networks and architecture of safety critical systems.

Dimitris Konstantinou received the Diploma in electrical and computer engineering from the Democritus University of Thrace, Xanthi, Greece, in 2016, where he is currently pursuing the Ph.D. degree. His research interests include low power system-on-chip architectures, and computer architecture.

Chrysostomos Nicopoulos received the B.S. and Ph.D. degrees in electrical engineering with a specialization in computer engineering from Pennsylvania State University, State College, PA, USA, in 2003 and 2007, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus. His current research interests include networks-on-chip, computer architecture, multi/many-core microprocessor and computer system design.

Giorgos Dimitrakopoulos received the B.S, M.Sc. and Ph.D. degrees in Computer Engineering from University of Patras, Patras, Greece, in 2001, 2003 and 2007, respectively. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece. He is interested in the design of digital integrated circuits, electronic design automation, and computer architecture, with emphasis in low-power systems design.