This document is a preprint of: V. Razilov, J. Zhong, E. Matus et al., “Dual Vector Load for Improved Pipelining in Vector Processors,” in Proceedings of IEEE Symposium on Low-Power and High-Speed Chips (COOLCHIPS 2023), Tokyo, Japan, (pp. 1-6), Apr 2023. DOI:10.1109/COOLCHIPS57690.2023.10121996

Dual Vector Load for Improved Pipelining in Vector Processors

Viktor Razilov*, Juncen Zhong*, Emil Matúš*, and Gerhard Fettweis*†
*Vodafone Chair Mobile Communications Systems, Technische Universität Dresden, Germany
†Barkhausen Institut, Dresden, Germany
{viktor.razilov, juncen.zhong, emil.matus, gerhard.fettweis}@tu-dresden.de

***Abstract*—Vector processors execute instructions that manipulate vectors of data items using time-division multiplexing (TDM). Chaining, the pipelined execution of vector instruction, ensures high performance and utilization. When two vectors are loaded sequentially to be the input of a follow-up compute instruction, which is often the case in vector applications, chaining cannot take effect during the duration of the entire first vector load. To close this gap, we propose dual load: A parallel or interleaved load of the two input vectors. We study this feature analytically and make statements on necessary conditions for performance improvements. Our investigation finds that compute-bound and some memory-bound applications profit from this feature when the memory and compute bandwidths are sufficiently high. A speedup of up to 33 % is possible in the ideal case. Our practical implementation shows improvements of up to 21 % with a hardware overhead of less than 2 %.**

***Index Terms*—RISC-V, dual load, vector processor, DSP.**

I. Introduction

In the quest for ever higher energy efficiency, computer architectures exploit, among other types of parallelism, data-level parallelism (DLP). As a major overhead in programmable architectures is the cost of fetching and decoding individual instructions, instruction set architectures (ISAs) have been extended with instructions that operate on vectors of data. Classic versions of these single-instruction multiple-data (SIMD) extensions, such as Intel’s streaming SIMD extensions (SSEs) or ARM’s “Neon” extension, follow the array processor paradigm [1]. They duplicate the functional units (FUs) in the datapath for concurrent processing of multiple data items. The resulting design is simple but constrains the vector length.

Vector processors execute SIMD instructions in a time-multiplexed manner instead, allowing the vector length to be larger than the number of FUs. In addition, they take advantage of instruction-level parallelism (ILP), as long-running vector instructions may run in parallel through a technique called chaining. In recent years, vector processors have found a renewed interest as scalable, flexible, and energy-efficient compute platforms [2], [3]. This trend is reflected by ISAs proposing vector computing extensions, e.g, ARM’s SVE [4] or the RISC-V “V” vector extension (RVV) [5]. Possible applications with significant DLP include, but are not limited to, deep neural networks [6], scientific computing [7], or communications signal processing [8].

The most common vector instructions perform operations on two or more operand vectors. When these need to be loaded from memory, as is often the case, the operation FUs stall during the entire loading time of the first vector. Only after initiation of the second load, chaining can take effect. Dual load of both operand vectors in parallel is able to act as remedy against this stall. It has long been a feature of digital signal processors (DSPs) [9] but are yet to be considered in the context of vector processors. We therefore propose dual load instructions as RVV extensions and investigate the performance benefit in a theoretical model and in an implementation.

The rest of the paper is structured as follows. Section II provides background information and a survey of related work. In Section III, we describe dual load instructions and their semantics. Section IV presents a theory for the conditions under which dual load is beneficial and the attained gain, while Section V presents an implementation based on a modified RVV processor and performance measurements. The paper is concluded in Section VI.

II. Background and Related Work

If a program applies identical operations on vectors of l_a independent data items it exhibits DLP. Vector processors, whose instruction operands are vector registers of l_v items held in a vector register file (VRF), are suited well for such applications. Neither l_v nor the throughput of the FUs is prescribed by the ISA and may be tuned by the designer for their particular use case.

To decrease vector loop iterations, l_v will often be larger than the consumption rate of the FUs. Instructions then occupy the execution stage for multiple cycles. Subsequent instructions with a data dependancy but no structural hazard are usually *chained* to start once their predecessor produced the first elements. This pipelining increases performance and utilization of the FUs and is thus implemented in most modern vector processors

A helpful model for a processor’s performance is the roofline model [10], [11]. It differentiates between the processor’s memory bandwidth β and compute bandwidth π. Depending on the algorithm’s operational intensity, that is the ratio

$$I = \frac{W}{Q} \qquad (1)$$

between the number of operations W and number of data items to be read or written Q, either of the bandwidths constraints

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This document is a preprint of: V. Razilov, J. Zhong, E. Matus et al., “Dual Vector Load for Improved Pipelining in Vector Processors,” in Proceedings of IEEE Symposium on Low-Power and High-Speed Chips (COOLCHIPS 2023), Tokyo, Japan, (pp. 1-6), Apr 2023. DOI:10.1109/COOLCHIPS57690.2023.10121996

the achievable performance to

$$\Pi = \min(\pi, I\beta). \tag{2}$$

In this model,

$$\iota = \frac{\pi}{\beta} \tag{3}$$

marks the “knee” of the roofline model. An application is memory-bound if its arithmetic intensity is below the knee and compute-bound otherwise. β and Q are usually measured in $\frac{\mathrm{B}}{\mathrm{cycle}}$ or B, respectively, but in this paper we abstract from the data item size and opt for the effective items per cycle or number of items, respectively. Generality is not lost.

Due to imperfections and stalls, a given algorithm will run with the actual performance P which will often be sub-optimal, i.e. below Π. The utilization of an architecture's available resources may then be inferred from

$$\eta = \frac{P}{\Pi}. \tag{4}$$

When applied to vector processors, β is given by the throughput of the load-store unit (LSU) and π by the effective throughput of the remaining arithmetic FUs, such as the arithmetic logic unit (ALU) or the multiplier (MUL).

Examples of recent vector processors implementing the RVV specification include but are not limited to Ara [12], [13], Sargantana [7], RISC-V[2] [14], Vicuna [15], Vitruvius+ [16], and unnamed others [17], [18]. All of these include only a single load-store unit (LSU) and can therefore execute only one memory instruction at a time. DSPs in contrast, often include a dual LSU to improve performance [19], [20]. Dual load has been brought from the DSP domain to a RISC SIMD processor in [9]. We extend this idea to vector processing by dual-loading entire vectors and provide a theoretical framework to estimate the performance gain.

III. Dual Vector Load

The execution pattern of an in-order single-issue vector processor is illustrated by Fig. 1 for element-wise multiplication of vectors. Two vectors of size $l_\mathrm{v} = 8$ are loaded from main memory, muliplied, and the result is then stored back. There are $Q = 3l_\mathrm{v}$ items to be read or written and $W = l_\mathrm{v}$ operations to be performed resulting in an arithmetic intensity of $I = \frac{1}{3}\,\frac{\mathrm{OP}}{\mathrm{item}}$. A processor with the compute bandwidth $\pi = 2\,\frac{\mathrm{OP}}{\mathrm{cycle}}$ and the memory bandwidth $\beta = 4\,\frac{\mathrm{item}}{\mathrm{cycle}}$ should execute this program with a memory-bound throughput of $\Pi = \frac{4}{3}\,\frac{\mathrm{OP}}{\mathrm{cycle}}$ according to the roofline model [10]. From now on, we will omit the unit of this units for brevity.

A vector processor with these parameters does not attain this bound as illustrated in Fig. 1a. The LSU is only utilized $\eta = 75\,\%$ of the cycles because it stalls when it waits for the results of the multiplication which starts only once the second load instruction has produced the first results. Interestingly, the roofline model would suggest to increase the memory bandwidth to speed up this memory-bound application, however increasing the compute bandwidth is more efficient here. To achieve full utilization, one could add two additional multipliers (c.f. Fig. 1b). However, this comes at a significant area cost.

We propose dual load instead. If the same memory bandwidth is shared between the two loading instructions, the multiplication can begin execution earlier (c.f. Fig. 1c). When loading is completed, enough items to be stored have already been produced by the MUL, so storing can be initiated right away. An LSU stall can be avoided this way without expensive additional multipliers.

Concurrent dual load needs support by the memory system, of course. Besides area-expensive dual-ported static random-access memory (SRAM), concurrent dual access can be implemented with banking or by placing the vectors into different memories at compile time. These approaches may be feasible for application-specific embedded systems.

In other cases, dual load may be executed in an interleaved fashion as in Fig. 1d. The LSU alternates between loading elements of the first and the second vector via a single memory port. To avoid latencies between the alternation, both vectors need to reside in the closest data cache (D$), which is realistic given the spatial locality exhibitied by many programs. In the Fig. 1 example, the performance gain of interleaved dual load is not as high as the one of concurrent dual load but still non-zero with less bespoke memory system requirements. Under certain conditions, interleaved dual load is even on par with concurent dual load as analyzed in Section IV.

Dual load may be exposed to the programmer as explicit dual vector load instructions. Figure 2 exemplifies an ISA extension for RVV [5] which supports two different addressing modes. The `vdl` instruction interprets both scalar source registers as two separate pointers to vectors, whereas the `vdol` instructions uses the second source source register as the offset of the second vector with respect to the first. Both instructions then load the vectors to two consecutive vector registers. The vector length is set with the `vset{i}vl{i}` beforehand, the data item width is encoded in the instruction itself as the *effective element width* (EEW) like in all other vector load instructions. The semantics of this extension are in line with the RVV specification and should therefore not incur siginifant complications to the decode and control logic of the vector processor. Even implicit additional destination registers is already part of RVV due to the segmented load instructions [8]. It is left to the designer whether to implement the dual vector load instructions in concurrent or interleaved fashion or whether to decompose it into two separate load instructions if dual load is not worthwhile.

Alternatively, dual vector load can be accomplished without special instructions by means of instruction fusion. If the processor detects two vector loads followed by a operation dependant on both, it can merge the two instructions into a concurrent or interleaved dual load. This approach can handle cases where one or both loads are strided or indexed, as well.

IV. Performance Analysis

Whether dual load is beneficial or not depends on variety of characteristics of the program and the vector processor. Figure 3 shows one counterexample where dual load does not

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This document is a preprint of: V. Razilov, J. Zhong, E. Matus et al., “Dual Vector Load for Improved Pipelining in Vector Processors,” in Proceedings of IEEE Symposium on Low-Power and High-Speed Chips (COOLCHIPS 2023), Tokyo, Japan, (pp. 1-6), Apr 2023. DOI:10.1109/COOLCHIPS57690.2023.10121996

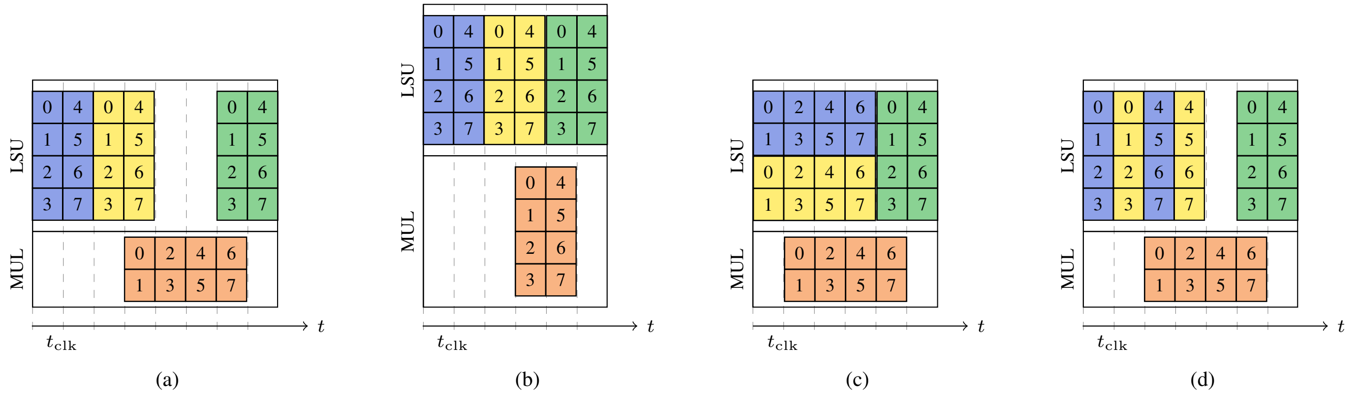

Fig. 1. Schematic execution patterns on different vector processors and as-late-as-possible (ALAP) scheduling. The vector processors are built out of different FUs represented by the different lanes of the diagram. Each square corresponds to a data item processed in a cycle with the number denoting the order. Every instruction is colored differently. (a) Vector processor without dual load, $\beta = 4$, and $\pi = 2$. (b) Vector processor without dual load, $\beta = 4$, and $\pi = 4$. (c) Vector processor with dual load, $\beta = 4$, and $\pi = 2$. (d) Vector processor with interleaved dual load, $\beta = 4$, and $\pi = 2$.

```
1  vdl<eew> vd, (rs1), (rs2) ; Load data at rs1 into vd, data at rs2 to vd + 1
2  vdol<eew> vd, (rs1), rs2 ; Load data at rs1 into vd, data at rs1 + rs2 to vd + 1
```

Fig. 2. Dual vector load ISA extension to RVV.

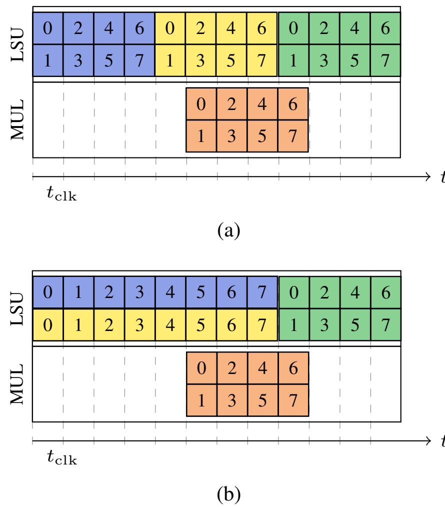

Fig. 3. Schematic execution pattern on vector processors with and without dual load. The colors denote the different executed instructions and the number the order of the vector elements. In this example, dual load does not bring any benefit. (a) Vector processor without dual load, $\beta = 2$, and $\pi = 2$. (b) Vector processor with dual load, $\beta = 2$, and $\pi = 2$.

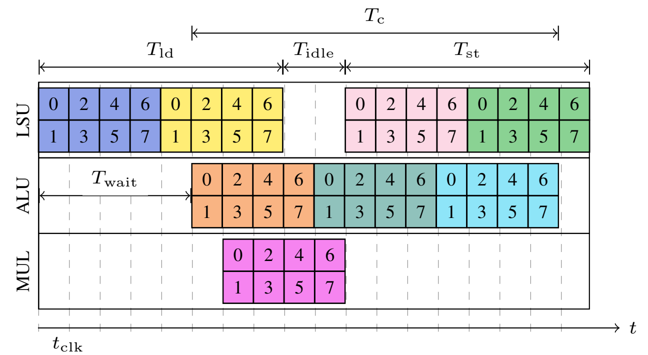

Fig. 4. Overview of the execution phases in a canonical vector program.

speed up the computation. In this section, we derive guidelines for a simple model of vector programs and processors. This should cover the most likely programs if not every edge case.

We assume a vector processor that issues one instruction per cycle without impairments or additional latencies. After decoding, the instruction is either executed in the LSU or in one of the compute FUs (e.g., ALU or MUL) where the instructions are executed with an effective throughput of β or π, respectively. All FUs read from or write in a single cycle to the VRF without contention. The interaction between the LSU and the D$ is also without impairment. Many of these assumptions are overly optimistic but simplify the analysis. We will later assess the impact of any degradations to this optimistic model.

A canonical vector program of our study is divided into three phases depicted in Fig. 4:

1) The loading phase, in which n_{ld} instructions read $Q_{\mathrm{r}} = n_{\mathrm{ld}} l_{\mathrm{v}}$ data items. Its duration is $T_{\mathrm{ld}} = \frac{Q_{\mathrm{r}}}{\beta}$.
2) The compute phase starts after T_{wait} assuming as-late-as-possible (ALAP) scheduling. In this phase, n_{c} effective instructions perform $W = n_{\mathrm{c}} l_{\mathrm{v}}$ operations. Not all compute instructions are considered here as they may be masked by instructions running on another FU in parallel. An example for a masked instruction is the MUL instruction in Fig. 4. The relevant intructions may be understood as the critical path of computation. Its duration is generally $T_{\mathrm{c}} = \frac{W}{\pi}$ but may be higher if the

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This document is a preprint of: V. Razilov, J. Zhong, E. Matus et al., "Dual Vector Load for Improved Pipelining in Vector Processors," in Proceedings of IEEE Symposium on Low-Power and High-Speed Chips (COOLCHIPS 2023), Tokyo, Japan, (pp. 1-6), Apr 2023. DOI:10.1109/COOLCHIPS57690.2023.10121996

critical path switches between FUs.

3) The storing phase, in which n_{st} instructions store $Q_{\mathrm{w}} = n_{\mathrm{st}} l_{\mathrm{v}}$ data items. Its duration is $T_{\mathrm{st}} = \frac{Q_{\mathrm{w}}}{\beta}$. T_{idle} denotes the time span between the loading and the storing phase where the LSU is idle.

Chaining leads to an overlap between the compute phase and the other two phases. Dual load aims to maximize the overlap by minimizing T_{wait} and T_{idle}.

Because dual load shares the memory bandwidth between two loads, it needs to be twice as high as the compute bandwidth to not throttle the compute FU:

$$\beta \geq 2\pi. \tag{5}$$

In other words, dual vector load exploits that not every operand item is consumed at once. One reason why dual load has no effect in the example of Fig. 3 is that (5) is not satisfied and T_{wait} not lowered. Compute instructions need to be long-running meaning that

$$\frac{l_{\mathrm{v}}}{\pi} > 1 \tag{6}$$

holds. Dual interleaved load additionally requires memory instructions to be long-running, as well.

A program may profit from dual load if it loads at least two vectors in the loading phase which are then used by the first (critical) instruction in the compute phase. Given our assumptions, the waiting time is then

$$T_{\mathrm{wait}} = \begin{cases} \frac{l_{\mathrm{v}}}{\beta} + 1 & \text{Single load} \\ 1 & \text{Dual load} \\ 2 & \text{Dual interleaved load.} \end{cases} \tag{7}$$

Additionally, the compute phase needs to be sufficiently large. Only if there is a

$$T_{\mathrm{idle}} > 0 \tag{8}$$

in the single-load case, can this T_{idle} be reduced by dual load. It follows that

$$T_{\mathrm{c}} > T_{\mathrm{ld}} + T_{\mathrm{st}} - 1 - T_{\mathrm{wait}}. \tag{9}$$

needs to be fulfilled which can be related to the arithmetic intensity: If it is higher than

$$I > \check{I} = \frac{\pi}{\beta}\left(1 - \frac{l_{\mathrm{v}} + 2\beta}{Q}\right) < \iota, \tag{10}$$

then there is speed-up by dual load. Among the applications adhering to the aforementioned canonical structure, all compute-bound applications and some memory-bound applications with an arithmetic intensity close to the knee of the roofline model satisfy this condition.

The gain of dual load will be

$$T_{\mathrm{gain}} = \begin{cases} \min(T_{\mathrm{idle}}, \frac{l_{\mathrm{v}}}{\beta}) & \text{Dual load} \\ \min(T_{\mathrm{idle}}, \frac{l_{\mathrm{v}}}{\beta} - 1) & \text{Dual interleaved load} \end{cases} \tag{11}$$

in this cases. The biggest relative gain is when $T_{\mathrm{idle}} = \frac{l_{\mathrm{v}}}{\beta}$ at an arithmetic intensity of

$$\hat{I} = \frac{\pi}{\beta}\left(1 - \frac{2\beta}{Q}\right) < \iota \tag{12}$$

TABLE I
INVESTIGATED VECTOR PROCESSOR CONFIGURATIONS

Name	π	β	$\frac{\pi}{\beta}$	l_{v}	VLEN
A	2	4	0.5	8	512
B	2	4	0.5	16	1024
C	4	8	0.5	16	1024
D	4	8	0.5	32	2048

in the dual load case. The corresponding speedup factor is given by

$$S = \frac{\frac{l_{\mathrm{v}}}{\beta} + T_{\mathrm{c}} + 2}{T_{\mathrm{ld}} + T_{\mathrm{st}}} = \frac{1 + \frac{Q}{l_{\mathrm{v}}}}{\frac{Q}{l_{\mathrm{v}}}}. \tag{13}$$

Because a canonical program consists of at least two loads and one store, Q is at least $Q \geq 3l_{\mathrm{v}}$ which implies that $S \leq \frac{4}{3}$ is the highest speedup factor in our model. One case where the maximum is achieved is illustrated in Fig. 1a and Fig. 1c. Dual interleaved load will perform slightly worse.

The latencies and stalls that were omitted so far will usually not significantly reduce the execution gap between the loading and storing phase and therefore the saving potential of dual load. Delays in one phase will propagate into the next phase because of dependencies most of the time. But they might still reduce the speedup and even counter the effects of dual load.

V. Experimental Results

As a proof-of-concept, we have added the dual load feature in the interleaved variant to a modified[1] version of the open-source RVV processor Ara [12], [13]. The undertaken implementation steps can be summarized as follows:

1) Increase the memory bandwidth to $\beta = 2\pi$, i.e., 128 bits per lane. This required and additional operand queue for loading, an additional operand requester for storing and a larger result queue in the vector load unit (VLDU). This step is not dual-load-specific as memory-bound kernels profit from it even when employing single load.
2) Modify the decoder to interpret the `vlsseg5e<eew>` instruction as dual load. The RISC-V opcodes left for custom instructions could have also been used.
3) Add a second address generation unit (AGU) to the VLDU. The addresses generated by the two AGUs are applied to the memory port in alternating order during the dual load process.

Of this processor we have selected the configurations A-D detailed in Table I. As we have implemented dual interleaved load, $\frac{l_{\mathrm{v}}}{\beta} > 1$ needs to be fulfilled, as well. On these configurations, we evaluate dual load with the example of the three vector kernels in Table II that represent the different cases analyzed in Section IV:

1) VOP1 is akin to the example in Fig. 1. Two vectors are loaded, added element-wise and the resulting vector stored back. The arithmetic intensity of this kernel is

[1] Compared to the original Ara, this in-house modification offers support for the segment load instruction defined by the RVV specification.

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This document is a preprint of: V. Razilov, J. Zhong, E. Matus et al., “Dual Vector Load for Improved Pipelining in Vector Processors,” in Proceedings of IEEE Symposium on Low-Power and High-Speed Chips (COOLCHIPS 2023), Tokyo, Japan, (pp. 1-6), Apr 2023. DOI:10.1109/COOLCHIPS57690.2023.10121996

TABLE II
OVERVIEW OF THE INVESTIGATED VECTOR KERNELS AND SOME OF THEIR PARAMETERS.

Kernel	Processor	W	Q	I	$\check{I}$	$\hat{I}$	S (Predicted)	S (Measured)
VOP1	A	l_v	$3l_v$	0.33	0.17	0.33	1.14	1.21
	B	l_v	$3l_v$	0.33	0.25	0.42	1.17	1.11
	C	l_v	$3l_v$	0.33	0.17	0.33	1.14	1.17
	D	l_v	$3l_v$	0.33	0.25	0.42	1.17	1.05
VCMUL	A	$4l_v$	$6l_v$	0.67	0.33	0.42	1.05	1.06
	B	$4l_v$	$6l_v$	0.67	0.38	0.46	1.09	1.03
	C	$4l_v$	$6l_v$	0.67	0.33	0.42	1.05	1.04
	D	$4l_v$	$6l_v$	0.67	0.38	0.46	1.09	1.03
VOP2	A	$2l_v$	$6l_v$	0.33	0.33	0.42	1.00	1.00
	B	$2l_v$	$6l_v$	0.33	0.38	0.46	1.00	0.99
	C	$2l_v$	$6l_v$	0.33	0.33	0.42	1.00	1.00
	D	$2l_v$	$6l_v$	0.33	0.38	0.46	1.00	0.98

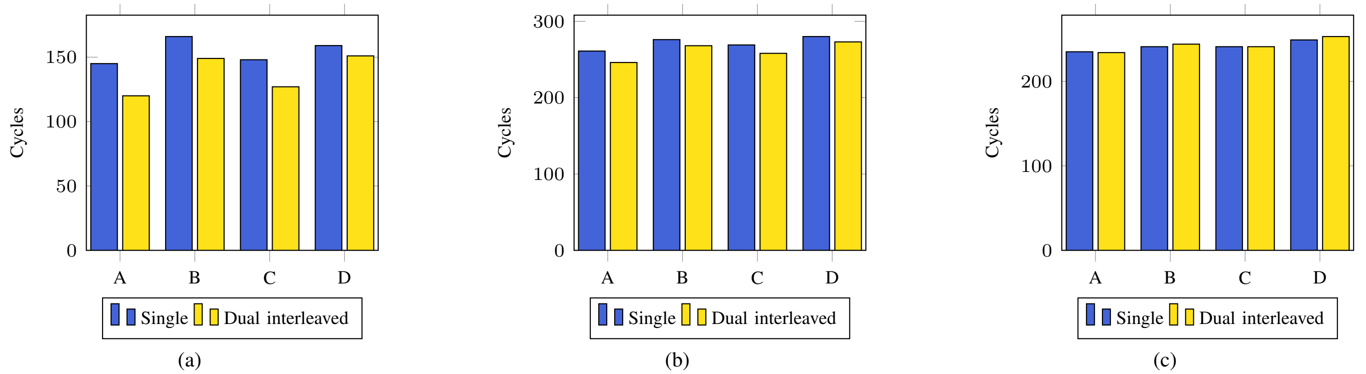

Fig. 5. Cycle count of the different kernels with single load and our dual interleaved load implementation. The apllication vector length is $l_a = 4l_v$. (a) VOP1. (b) VCMUL. (c) VOP2.

$I = \hat{I}$ for some processor configurations and (parallel) dual load can achieve the greatest speedup here.

2) VCMUL performs an element-wise multiplication of two complex vectors implemented as four real vectors. These are combined using four multiplications and stored back to memory as two vectors. Some communications signal processing tasks have a similar structure [8]. This kernel is compute-bound and its arithmetic intensity above the identified lower bound $I > \check{I}$. The model in Section IV suggest a small speed up here.
3) VOP2 consists out of two independent vector additions in the vector loop: $\vec{c} = \vec{a} + \vec{b}$, $\vec{f} = \vec{d} + \vec{e}$. The arithmetic intensity is lower than the required minimum $I < \check{I}$ and indeed, the load of $\vec{d}$ and $\vec{e}$ can be scheduled to fill the LSU idle time in VOP1. Dual load is thus futile.

The application vector length was fixed to $l_a = 4l_v$ for all of the cases resulting in four iterations of the vector loop. The width of single data item is 64 bit in all cases.

Figure 5 depicts the measured cycle count when using single and dual interleaved load in the test cases. As expected, VOP1 receives the greatest performance boost from dual load which goes up to 21 %. VCMUL benefits a little, while VOP2 performance is the same or even slightly lower. Deviations from the predicted gain are caused by the various latencies and stalls that were ommited in the model to make it analytically manageable. These degradations lower the actual speedup in most cases and lead to a subpar utilization in all recored cases. VRF contentions, which are dependent on VRF implementation details and on the vector register selection in the assembly code, stand out, in particular. Another microarchitecture may resolve some of these issues and bring dual load performance gain closer to the ideal.

The results of the synthesis on a Xilinx Virtex 7 FPGA are summarized in Table III. An instance of Ara with 2 lanes, a memory bandwidth of 256 bit cycle, and a VLEN of 1024 bits increases in size by roughly 2 % when the dual load feature is added. The area overhead will be less pronounced in a complete sytem with the scalar base core and on-chip memory. Building a full-fledged out-of-order vector processor comes at a significantly higher area and power cost.

VI. CONCLUSION

In this paper, we have proposed dual vector load as a means to improve performance for the class of vector applications that start with binary vector operations on input vectors. The idea is to share the memory bandwidth between the loads of two input vectors such that the followup compute instruction can begin earlier. Dual load can be implemented concurrently

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

This document is a preprint of: V. Razilov, J. Zhong, E. Matus et al., “Dual Vector Load for Improved Pipelining in Vector Processors,” in Proceedings of IEEE Symposium on Low-Power and High-Speed Chips (COOLCHIPS 2023), Tokyo, Japan, (pp. 1-6), Apr 2023. DOI:10.1109/COOLCHIPS57690.2023.10121996

TABLE III
SYNTHESIS RESULTS FOR ARA WITH 2 LANES AND A VLEN OF 1024 BIT WITHOUT AND WITH DUAL LOAD ON A XILINX VIRTEX 7 AND A CLOCK FREQUENCY OF 50 MHZ.

Module	w/o dual load		w/ dual load	
	Slice LUTs [k]	Slice Registers [k]	Slice LUTs [k]	Slice Registers [k]
Ara	140	32.9	143	33.3
VLSU	13.3	2.26	15.7	2.56
VLDU	9.63	0.87	11.2	0.96
AGU	2.26	0.62	3.04	0.83
VSTU	1.12	0.40	1.17	0.40
Other	0.33	0.38	0.27	0.38
Lane 0	54.1	14.1	54.2	14.1
Lane 1	53.4	14.1	53.8	14.1
Other	19.4	2.50	19.6	2.51

or interleaved. The former is faster but the latter may be easier to implement in general-purpose processors.

When certain conditions are met, a speedup can be achieved without an area- and power-hungry increase of compute or memory bandwidth. We have derived analytic statements on these conditions based on an idealized model of a vector processors. The memory bandwidth needs to be twice as high as the compute bandwidth, compute instructions need to be long running and the arithmetic intensity needs to be above a certain threshold that is slightly below the “knee” of the roofline model. Dual load can achieve a performance gain of up to 33 % with only a minor increase in area.

We have presented a first implementation as an extension to an RVV processor and demonstrated a cycle count reduction of up to 21 %. Degradations in the vector processor generally reduced the observed performance improvement. Nonetheless, the results tend to confirm the model’s qualitative findings. Future work will address further optimizations to bring the performance closer to the theoretical bound.

ACKNOWLEDGMENT

Funded in part by the European Union in the project “COREnext” (Grant Agreement number 101092598). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. Additionally, this work was funded in part by the German Federal Ministry of Education and Research (BMBF) in the project “E4C” (project number 16ME0426K).

REFERENCES

[1] M. J. Flynn, “Some computer organizations and their effectiveness,” *IEEE Trans. Comput.*, vol. C-21, no. 9, pp. 948–960, Sep. 1972.

[2] S. F. Beldianu and S. G. Ziavras, “Performance-energy optimizations for shared vector accelerators in multicores,” *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 805–817, Mar. 2015.

[3] D. Dabbelt, C. Schmidt, E. Love, H. Mao, S. Karandikar, and K. Asanovic, “Vector processors for energy-efficient embedded systems,” in *3rd ACM Int. Workshop Many-core Embedded Systems*, Seoul, Republic of Korea, Jun. 2016, pp. 10–16.

[4] N. Stephens *et al.*, “The ARM scalable vector extension,” *IEEE Micro*, vol. 37, no. 2, 2017, pp. 26–39, Mar./Apr. 2017.

[5] *RISC-V “V” Vector Extension*, Version 1.0, Sep. 2021. [Online]. Available: https://github.com/riscv/riscv-v-spec/releases/download/v1.0/riscv-v-spec-1.0.pdf

[6] S. Di Mascio, A. Menicucci, E. Gill, G. Furano, and C. Monteleone, “On-board decision making in space with deep neural networks and RISC-V vector processors,” *J. Aerosp. Inf. Syst.*, vol. 18, no. 8, pp. 553–570, Aug. 2021.

[7] V. Soria-Pardos *et al.*, “Sargantana: a 1 GHz+ in-order RISC-V processor with SIMD vector extensions in 22nm FD-SOI,” in *25th Euromicro Conf. Digit. Syst. Design (DSD)*, Maspalomas, Spain, Aug. 2022, pp. 254–261.

[8] V. Razilov, E. Matúš, and G. Fettweis, “Communications signal processing using RISC-V vector extension,” in *IEEE Int. Wireless Communications and Mobile Computing Conf. (IWCMC)*, Dubrovnik, Croatia, Jun. 2022, pp. 690–695.

[9] H. Shi, “RISC + SIMD = DSP?,” *IEEE Int. Conf. Acoustics, Speech, Signal Proc. (ICASSP)*, Istanbul, Turkey, pp. 3211–3214, Jun. 2000.

[10] S. Williams, A. Waterman, and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” *Commun. ACM*, vol. 52, no. 4, Apr. 2009, pp. 65–76.

[11] G. Ofenbeck, R. Steinmann, V. Caparros, D. G. Spampinato, M. Püschel, “Applying the roofline model,” in *IEEE Int. Symp. Perf. Anal. Syst. Softw. (ISPASS)*, Monterey, CA, USA, Mar. 2014, pp. 75–85.

[12] M. Cavalcante, F. Schuiki, F. Zaruba, M. Schaffner, and L. Benini, “Ara: a 1-GHz+ scalable and energy-efficient RISC-V vector processor with multiprecision floating-point support in 22-nm FD-SOI,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* , vol. 28, no. 2, pp. 530–543, Feb. 2020.

[13] M. Perotti, M. Cavalcante, N. Wistoff, R. Andri, L. Cavigelli, and L. Benini, “A ‘new Ara’ for vector computing: an open source highly efficient RISC-V V 1.0 vector processor design,” in *IEEE 33rd Int. Conf. Appl.-Specif. Syst. Archit. Process. (ASAP)*, Gothenburg, Sweden, Jul. 2022, pp. 43–51.

[14] K. Patsidis, C. Nicopoulos, G. C. Sirakoulis, and G. Dimitrakopoulos, “RISC-V^2: a scalable RISC-V vector processor,” in *IEEE Int. Symp. Circuits and Systems (ISCAS)*, Seville, Spain, Oct. 2020, pp. 1–5.

[15] M. Platzer and P. Puschner, “Vicuna: a timing-predictable RISC-V vector coprocessor for scalable parallel computation,” in *33rd Euromicro Conference Real-Time Systems (ECRTS)*, Dagstuhl, Germany, Jun. 2021, pp. 1:1–1:18.

[16] F. Minervini *et al.*, “Vitruvius+: an area-efficient RISC-V decoupled vector coprocessor for high performance computing applications,” *ACM Trans. Archit. Code Optim.*, early access, Dec. 2022, doi: 10.1145/3575861.

[17] M. Johns and T. J. Kazmierski, “A minimal RISC-V vector processor for embedded systems,” in *2020 Forum Specification and Design Languages (FDL)*, Kiel, Germany, Sep. 2020, pp. 1–4.

[18] C. Schmidt *et al.*, “An eight-core 1.44-GHz RISC-V vector processor in 16-nm FinFET,” *IEEE J. Solid-State Circuits*, vol. 57, no. 1, Jan. 2022, pp. 140–152.

[19] Qualcomm, “Hexagon SDK - DSP Processor,” Qualcomm Developer Network. https://developer.qualcomm.com/software/hexagon-dsp-sdk/dsp-processor (accessed Feb. 3, 2023).

[20] G. Wilson, “High-Performance DSP and Control Processing for Complex 5G Requirements,” Synopsys. https://www.synopsys.com/designware-ip/technical-bulletin/high-performance-dsp-for-5g-dwtb-q418.html (accessed Feb. 3, 2023).

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.