RISC-V 向量擴展指令實作

洪明祺
國立陽明交通大學
資訊工程學系
melany30031@gmail.com

***Abstract*—近年來向量擴展對於加速數據的平行運算像是機器學習、多媒體、數據流等領域加速變得越來越重要，尤其在於資源受限與功率有限的設備上。此專題是在一個 5-stage RISC-V 處理器中增加支援一部份向量指令，專題依照 RISC-V 規格書[1]、[2]，且增加的指令符合 RISC-V 向量擴展 v1.0 的微架構設計[3]。最後簡易測試純量與向量性能比較。**

Keywords—RISC-V; Vector extensions ; Vector accelerating;

I. INTRODUCTION

最近幾年，隨著人工智慧蓬勃發展，高度先進的機器學習算法適用於眾多領域，因而在日常應用中，機器學習算法被廣泛而大規模採用。而神經網路(Neural Network, NN)在語音識別、圖像[4]、模式[5]等應用中被認為是最先進、為機器學習中最常被使用的算法之一。隨著神經網路的出現，許多工程學科也出現了重大變化。然而由於神經網路的快速發展，對於各種硬體架構：如芯片設計，便受到了嚴重影響，隨著大量資料餵入模型訓練、工作負載增加，單位時間內需要更多的操作，神經網絡架構的計算精度不斷降低[6]、[7]，包括較大類型的自定義表示[8]、[9]。

許多研究都試圖尋找解決辦法，以解決計算增加帶來的效能不足問題：例如傳統通用處理器致力於指令層級平行 (Instruction-level parallelism, ILP)，雖然在指令集添加了單指令流多資料流(Single Instruction Multiple Data, SIMD)指令部分利用了數據級並行性(Data-Level Parallelism, DLP)，但吞吐量還是有限[10]、或是改良演算法以較大的單位模塊壓縮數據，減少所需的數據量[11]，但這些方法隨著數據資料越來越龐大，往往在很短的時間內就不再那麼的適用。

在對於大量數據的運用下追尋高效率、低耗能的同時，向量處理器也不失為一個優秀的解決方式，向量架構在有效結合高能源效率、高可編程性、高計算吞吐量方面幾乎是獨佔鰲頭的[12]、[13]。而在過去幾年中，RISC-V 已成為發展成熟、現代且開源免費的指令集架構(Instruction Set Architecture, ISA) 替代方案。因而造成了一波實作熱潮，並引發了新穎、定制化的 ISA 擴展及潛在標準化與微架構影響的大眾討論。

向量指令可以操作可變大小的不同向量，長度和元素大小可以在運行時設置。此外，單一向量指令會執行向量所有元素的計算，而 RISC-V 向量擴展更加靈活，可以使用 vsetvl 指令，在執行時動態調整寄存器大小、組數、元素數據寬度，以及掩碼操作等等。優點為解決了對未來的兼容性，並且讓同一份可執行文件，在不同架構的向量處理器中可充分利用各種不同寬度的向量處理器，執行一份代碼，兼容不同硬體架構，從而大大降低了軟體的維護成本。相比於傳統的 SIMD 技術，向量計算是一種硬體軟體更加解耦的技術，對於工程師更加友善、是一種軟硬體協同的技術典範。由於指令集限定了數據操作位寬，每次硬體對並行度的擴展都意代表著代碼的重寫與指令集的擴展，這將需要付出更多時間精力，對開發人員也更不友善。

II. HISTORY DEVELOPMENT

Cray 創立了 Cray Research，並在西元 1976 年推出了 CRAY-1。CRAY-1 使用向量寄存器架構以大幅降低啟動成本、支援非單元化跨度(nonunit stride)、發明鏈接，並開始使用向量指令以提高性能，減少指令編碼、地址計算和指令緩存缺失(cache miss)。

向量指令經過低潮期後於 1997 年重新出現 ：MMX 指令集，MMX 被添加到 x86 架構和 Intel Pentium 處理器中。指令是 SIMD，但向量只有 64 位元、8 個寄存器，並且只支援整數運算而沒有浮點數。一條指令可以同時執行兩個 32 位整數、四個 16 位整數或八個 8 位整數。

1999 的 SSE(Streaming SIMD Extension) 添加了 8 個新的 128 位寄存器和浮點運算。2008 年的進階向量擴展 (Advanced Vector Extensions, AVX)，SSE 寄存器從 128 位元增加到 256 位元，並改名成 AVX。寄存器從 8 個添加到 16 個。而 AVX 引入了三操作數指令格式，允許保存輸入寄存器。之後，AVX2 (2013) 添加了整數 256 位 的 SIMD 指令和浮點融合乘加。

多種現代 CPU 架構被設計為向量處理器。RISC-V 向量擴展遵循與早期向量處理器類似原理，並且在許多商業產品中實現。

III. RELATED WORK

在 RISC-V 越來越盛行的同時，除了 RV32I base 外，也有越來越多向量擴展的應用，例如開關電容 DC-DC 間轉換[15]，其集成的 SC(Switch Capacitor) DC-DC 轉換器為帶有向量加速器的 RISC-V 微處理器供電，從而實現改進的動態時鐘頻率調整(Dynamic voltage and frequency scaling, DVFS)，提升效能。其中關於 RISC-V 向量加速是以 64 位元向量加速器通過分攤數據並行操作的指令，獲取和控制開銷來提高能源效率。

此為資訊工程專題專題報告改寫，指導教授蔡淳仁

另一例子為通訊與訊號處理[16]，通訊信號處理與 DLP 有關，因其流程需要計算與邏輯操作，而 RISC-V 向量擴展則可以以單一指令執行，否則將需額外浪費多個基本指令。而 RISC-V 向量擴展的兼容性，則對於底層硬體平台較無關係，而若以 ARM 的可伸縮向量擴展 (Scalable Vector Extension, SVE) 使用循環預測來達到類似的效果，但相關耗費較大[17]。

至於標量處理器跟向量處理器，雖然向量處理器架構較為複雜、需要較多內存、成本較為昂貴，但是在數據密集型運算下，一行指令可執行許多操作，因而可節省許多程式碼所耗費的時間與空間提高效率 [18]。在圖形處理、人工智慧、網路安全等多領據皆適用。而相較於 ARM、MIPS 等，RISC-V 架構具有低功耗、低成本、開源開放、可模組化、簡潔、面積小和速度快等優點。

IV. ARHITECTURE

此專題為五流水線架構，分別為 IF(instruction fetch)、ID(decode) 、 EXE(execute) 、 MEM(memory) 、 WB(write back)，五個階段模組，還有計算執行到哪行指令(pc 值)的程式計數器(program counter, PC)、暫存 x1-x31 的 純量暫存器(scalar register)，其中 x0 固定為 0、執行運算的算術邏輯單元(Arithmetic logic unit, ALU)、暫存 v0-v31 的 向量暫存器(vector register)、儲存指令的指令記憶體(instruction memory) 跟儲存資料的記憶體(data memory)。如圖一。

A. Data Format

我目前所有指令為 32 位元，純量暫存器一組也為 32 位元、向量暫存器大小為 32*8 位元，也就是每個向量有 8 個元素、每個元素皆為 32 位元。相關格式如圖二，因這份專題主要為實作向量指令以硬體模擬加快課程上使用軟體執行的部分，需計算的資料較為單一，因此先預設了相關參數，節省為判斷輸入資料格式等額外資源耗費。

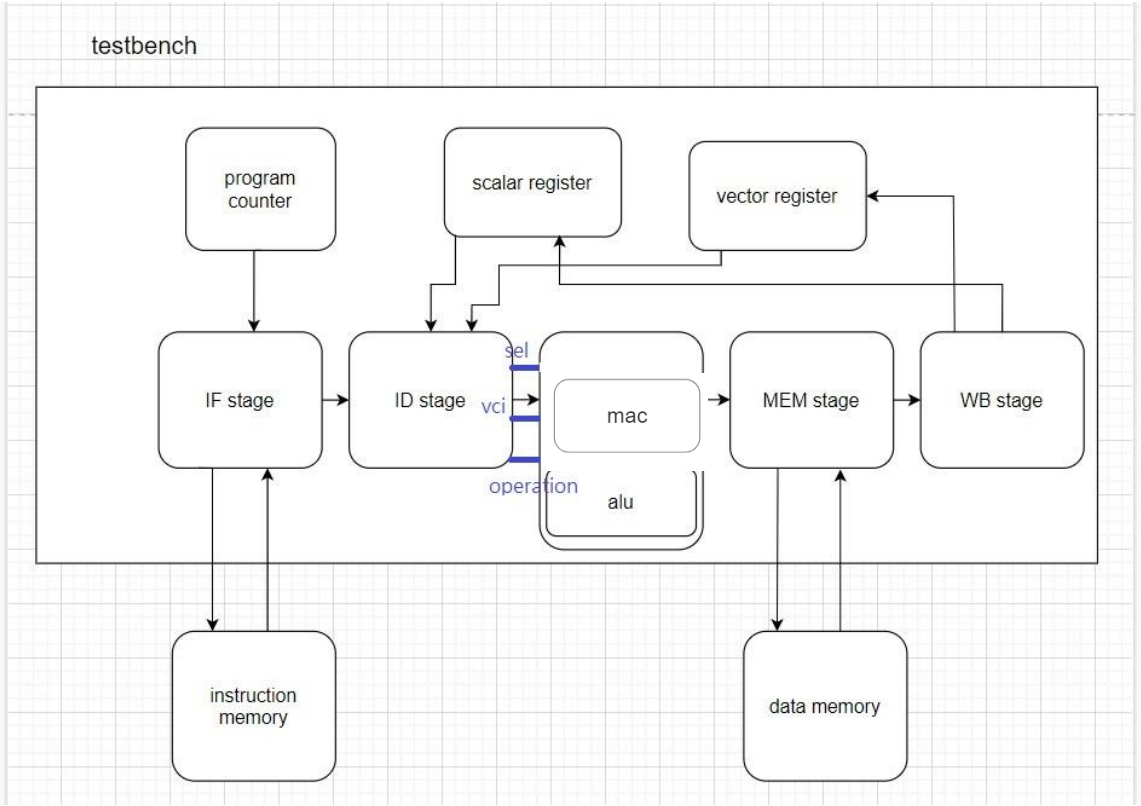
1). vill = 0 // 1 為指令異常，後續皆為 0

2). vma = 0 // 0 為 undisturbed，因流水線為順序執行 (in-order) 所以目的的暫存器保留原來的值即可

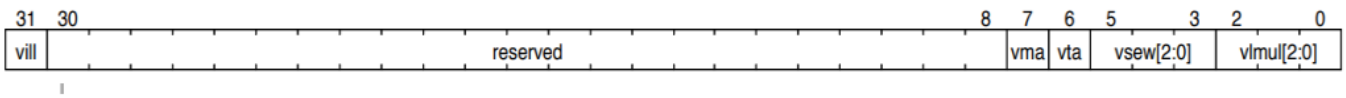
3). vta = 0 // 同上為 undisturbed

4). vsew = 010 // 向量元素的大小 010 為 32 位元

5). vlmul = 000 // 向量暫存組，為一向量指令可以操作多少向量



圖一、主要架構



圖二、向量格式

B. Support Instructions

因為簡易版的 RISC-V 向量擴充，目前除了 32 位元載入(load)、儲存(store)，以向量跟向量、向量跟純量、向量跟內存單元(immediate, imm)的加減法與左移右移指令與成加指令為主。

1). addi rd, rs1, imm // rd = rs1 + imm

2). vle32.v vd, (rs1) // vector load

3). vse32.v vs3, (rs1) // vector store

4). vadd.vv vd, vs2, vs1 // vs1 + vs2 = vd

5). vadd.vx vd, vs2, rs1 // vd[i] = rs1 + vs2[i]

6). vadd.vi vd, vs2, imm // vd[i] = imm + vs2[i]

7). vsub.vv vd, vs2, vs1 // vd = vs1 - vs2

8). vsub.vx vd, vs2, rs1 // vd[i] = rs1 - vs2[i]

9). vslide1up.vx vd, vs2, rs1

// vd[0]=x[rs1], vd[i+1] = vs2[i]

10). vslide1down.vx vd, vs2, rs1

// vd[i] = vs2[i+1], vd[vl-1]=x[rs1]

11). vmacc.vv vd, vs1, vs2

// vd[i] = +(vs1[i] * vs2[i]) + vd[i]

C. Implementation

1) **Instruction fetch：**

我設了 5 個流水現階段，其中因指令資料與數據資料分別放在外部記憶體，因此若要存取會耗費更多的時間單位而造成整個流水線停止，我設計是當每個階段執行完自己的指令時會停止直到每個階段執行完成，再同時移到下一流水階段，因此在挪移到下一個階段同時 stall 訊號會設成 0，這個時間單位主要執行資料傳輸與運算，之後會一直暫停直到記憶體 都存取完，在模擬時因對於外部記憶體有分別 enable 跟 ready 訊號以確定雙方資料接收時間的正確。

2) **Data memory access：**

而至於數據資料記憶體的部分，因傳輸資料為大小為 32 位元，因而無法一次傳輸完所有資料，因一個向量有 8 個元素，所以需要傳輸成功 8 次才能正確存取整個向量。相關操作類似指令記憶體，除了 stall 訊號直到存取完 8 個元素。我設 count 訊號為計算與判斷是否存取結束的依據，當 count 訊號為 7 接著下一個元素存取成功時，也就是最後一個元素。所以存取完成訊號會拉起來、count 訊號會重新設為 0 為預備下一次的數據資料存取。而若存取的是向量資料，因暫停時間一定比取出指令還要長，所以結束後就代表所有階段皆完成了工作，可以直接進入到下一個階段。

3) **PC counting：**

我設計的架構在沒有指令讀進來時，也就是最一開始整個會維持在暫停狀態，因此在計算 pc 值時，若當 stall 訊號一降下來就執行加 4 會造成第一個指令(instruction[0])被跳過的情況，所以為了解決這個問題，不同於其他部份都是 stall 訊號降下來時開始執行，program counter 中開始執行的識別訊號我設為當指令從記憶體提取成功的訊號，所以一旦完成指令提取才會更新下一個 pc 值，時間的部分因 stall 訊號降下來的條件為指令提取完成與同一時間並未執行數據資料存取，所以不會造成錯誤。

4) **Decode vector signal：**

關於向量相關指令，先在 decorder 完成對於指令的初步解碼，以通知後續階段須執行的訊號，跟 RISC-V base 比，多了：

1. sel : 取 opcode 初步判斷是要執行載入、儲存、base 的加法(addi)、或是向量的計算。
2. operation : 由 funct6 判斷是要執行加法、減法、抑或是左移一個元素或是右移，並傳到 alu。
3. vci : 由 funct3 判斷此指令是向量跟向量的、向量跟純量、或是向量跟 Immediate 計算。

而因目前支援指令較少，為追求效率，只針對不同的地方做解碼，如 vse32、vle32 直接取 opcode 判斷即可，因此就不多從 funct3、funct6 判斷以節省資源，而像是 vadd 因有支援 OPIVI、OPIVX、OPIVV 等較為細部的指令，除了 opcode 還須從 funct3、funct6 判斷。 表一為指令格式。

表一、指令格式

Instruction	*Funct6*	*Funct3*	*opcode*
vle32.v	X	000	0000111
vse32.v	X	000	0100111
vadd.vv	000000	000	1010111
vadd.vx	000000	100	1010111
vadd.vi	000000	011	1010111
vsub.vv	000010	000	1010111
vsub.vx	000010	100	1010111
vslide1up	001110	110	1010111
vslide1down	001111	110	1010111
vmacc	101101	010	1010111

5) **Vector operation at execute stage：**

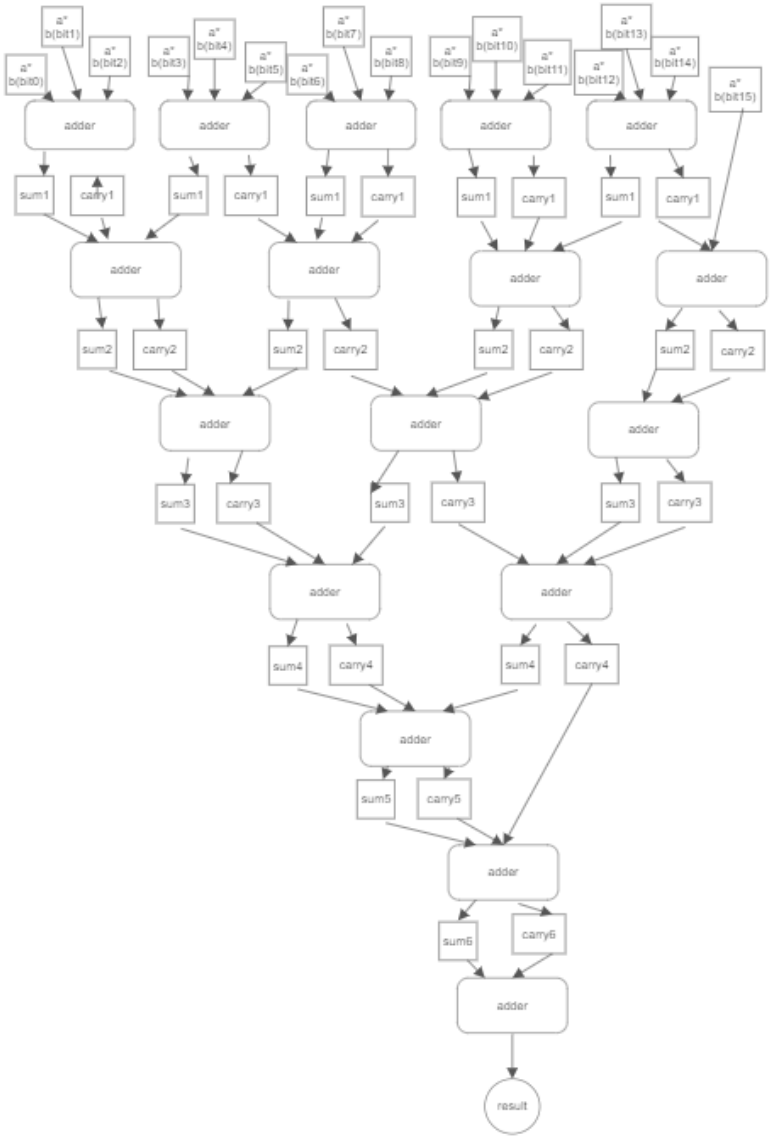
在 execute 階段，會從 decode 傳來的訊號中判斷這個指令是屬於向量的還是 RISC-V base 的，生成 is_vector、is_scalar 訊號，以告知 alu 關於計算的輸

入選擇與之後挪移到 write back 階段對於哪一個暫存器寫入的選擇，而若執行指令為乘加指令則從 mac 模組執行。因為指令有支援向量跟向量、向量跟純量、向量跟 immediate，所以同樣是向量的運算，也要做輸入判斷。我從圖一中的 vci 去判斷選擇 vs1、rs1、imm 哪一個輸入，其中若是後兩者，需填充(padding)成完整的向量以執行向量運算。rs1 是向量所有元素都填充成 rs1、而因為 immediate 只 有 5 個位元，若是負數的話直接填充 0 會造成錯誤。因此我先做了符號擴展(sign extension)，若是負數的話，最高位為 1，則位元 32 到位元 6 皆補 1，反之全補 0。符號擴展之後就如同 rs1 做填充。

6) **MAC：**

主要由 Wallace Tree 乘法器組成。核心概念為三組數據相加產生的兩個輸出：和(sum)與進位(carry)，在每三個為一組繼續相加，直到最後加總完。採用壓縮比為 3:2 的 Wallace Tree 乘法器的乘法延遲時間正比於 $\log_{3/2}\left(\frac{n}{2}\right)$。雖其缺點為不規則，然而在此專題中因輸入的資料格式皆固定，因而可直接設置其所需線路(如下圖三)，所以選擇此種乘法器實作。

而此專題模擬所餵入的資料數值並不會太大，因而模擬時設置在 16 位元內，所以不會有溢位(overflow)的情況，如同上述理由，輸入的資料皆為 16 位元。其中圖三中最上一層的數值，是以被乘數與乘數的每一位元做邏輯 AND (&)運算並同時從 16 位元補至 32 位元，以加快運算。



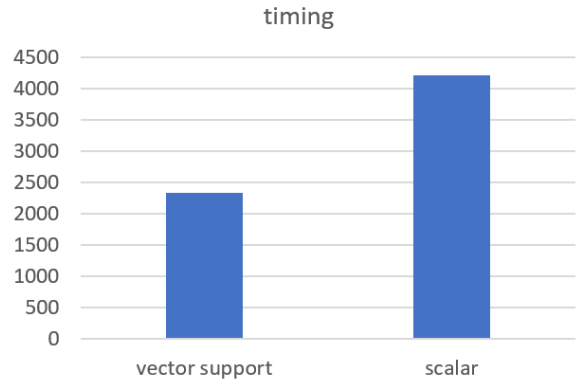
圖三、wallace tree 架構

7) **Write back data choosing：**

會由從 decode 階段挪移過來的 sel 訊號判斷寫入的資料是從 execute 階段計算出抑或是數據資料記憶體讀出來的。

V. RESULTS

模擬結束後所有耗費的時間單位為 2335，而若是都用純量一筆一筆計算與讀取資料， 計算方面像是 vadd、vsub、vslide1up、vslide1down 等指令皆須乘以向量中元素個數的指令個數時間，每個指令至少需額外 7 條指令才能完成相同工作，而 vmacc 指令還要再多加指令個數的加法指令，這還是不考慮有額外指令做為記錄每個向量起始元素儲存位址的情況，考慮到因模擬指令數量較少，相較之下 vle32.v、vse32.v 等指令相對於全純量存取需大量時間的指令占比不低，若計算量更高，差距會更為明顯，如下圖四。



圖四、向量指令與全部非向量運算時間

VI. CONCLUSION

因向量擴展十分繁雜，所以我這份專題只有支援部分指令，然而在某些較為簡易的應用時反而可以節省許多狀態判斷的時間與線路，提升效率。未來方向將以增多支援指令、以能處理更為全面的情況為目標。這份專題對我來說主要訓練為嘗試閱讀規格書[1]、[2]、[3]，在規劃與實作的過程中我學到了很多寶貴的經驗。

REFERENCES

[1] “The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213”, Editors Andrew Waterman and Krste Asanovic, RISC-V Foundation, December 2019.

[2] “The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203”, Editors Andrew Waterman, Krste Asanovi´c, and John Hauser, RISC-V International, December 2021.

[3] “RISC-V "V" Vector Extension Version 1.0” https://github.com/riscv/riscv-v-spec

[4] Sanjith M Gowda, B Rajeshwari, B Bajarangbali, "Minimalistic Vector Extension for Image Detection on Edge Nodes Using ‘VGG16’", 2022 International Conference of Science and Information Technology in Smart Administration (ICSINTESA), pp.6-11, 2022.

[5] Viktor Razilov, Juncen Zhong, Emil Matúš, Gerhard Fettweis, "Dual Vector Load for Improved Pipelining in Vector Processors", 2023 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), pp.1-6, 2023.

[6] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, “An always-on 3.8μJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28 nm CMOS,” in IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers, Feb. 2018, pp. 222–224.

[7] X. Sun et al., “Ultra-low precision 4-bit training of deep neural networks,” in Advances in Neural Information Processing Systems, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Red Hook, NY, USA: Curran Associates, 2020, pp. 1796–1807.

[8] S. Wang and P. Kanwar. (2019). Bfloat16: The Secret to High Performance on Cloud TPUS. [Online]. Available: https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-highperformance-on-cloud-tpus.

[9] P. Kharya. (May 2020). Tensorfloat-32 in the a100 GPU Accerlerates AI Training, HPC to 20X. [Online]. Available: https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/.

[10] Y. Lee, R. Avizienis, A. Bishara, R. Xia, D. Lockhart, C. Batten, and K. Asanovic, “Exploring the tradeoffs between programmability and ´ efficiency in data-parallel accelerators,” in International Symposium on Computer Architecture (ISCA), June 2011, pp. 129–140.

[11] Y. -K. Weng, S. -H. Huang and H. -Y. Kao, "Block-Based Compression for Reducing Indexing Cost of DNN Accelerators," 2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Penghu, Taiwan, 2021, pp. 1-2, doi: 10.1109/ICCE-TW52618.2021.9603025.

[12] J. Wawrzynek, K. Asanovic, B. Kingsbury, D. Johnson, J. Beck, and N. Morgan, “Spert-ii: a vector microprocessor system,” Computer, vol. 29, no. 3, pp. 79–86, March 1996.

[13] R. Espasa, F. Ardanaz, J. Emer, S. Felix, J. Gago, R. Gramunt, I. Hernandez, T. Juan, G. Lowney, M. Mattina, and A. Seznec, “Tarantula: a vector extension to the alpha architecture,” in Proc. International Symposium on Computer Architecture (ISCA), May 2002, pp. 281–292.

[14] John L. Hennessy and David A. Patterson. Computer Architectures: A Quantitative Approach. Morgan Kaufmann Publishers, Inc., second edition, 1996.

[15] B. Zimmer et al., "A RISC-V Vector Processor With Simultaneous-Switching Switched- Capacitor DC–DC Converters in 28 nm FDSOI," in IEEE Journal of Solid-State Circuits, vol. 51 , no. 4, pp. 930-942, April 2016, doi: 10.1109/JSSC.2016.2519386.

[16] V. Razilov, E. Matúš and G. Fettweis, "Communications Signal Processing Using RISC-V Vector Extension," 2022 International Wireless Communications and Mobile Computing (IWCMC), Dubrovnik, Croatia, 2022, pp. 690-695, doi: 10.1109/IWCMC55113.2022.9824961.

[17] A. Pohl, M. Greese, B. Cosenza and B. Juurlink, "A performance analysis of vector length agnostic code", Int. Conf. High Performance Computing & Simulation (HPCS), pp. 159-164, Jul. 2019.

[18] Mike Ashworth and Ian J. Bush and Martyn F. Guest ,” Vector vs. Scalar Processors: A Performance Comparison Using a Set of Computational Science Benchmarks” Ashworth2005VectorVS, 2005.