VIA: A Smart Scratchpad for Vector Units with Application to Sparse Matrix Computations

Julián Pavón[1,2], Osman Unsal[1] and Adrian Cristal[1,2]
[1]*Barcelona Supercomputing Center*
firstname.lastname@bsc.es
[2]*Universitat Politècnica de Catalunya*
firstname.lastname@upc.edu
Barcelona, Spain

***Abstract*—Sparse matrix operations are critical kernels in multiple application domains such as High Performance Computing, artificial intelligence and big data. Vector processing is widely used to improve performance on mathematical kernels with dense matrices. Unfortunately, existing vector architectures do not cope well with sparse matrix computations, achieving much lower performance in comparison with their dense counterparts.**

To overcome this limitation, we present the Vector Indexed Architecture (VIA), a novel hardware vector architecture that accelerates applications with irregular memory access patterns such as sparse matrix computations. There are two main bottlenecks when computing with sparse matrices: irregular memory accesses and index matching. VIA addresses these two bottlenecks with a smart scratchpad that is tightly coupled to the Vector Functional Units within the core. As a result, VIA achieves significant performance speedup over highly optimized state-of-the-art C++ algebra libraries. On average, VIA outperforms sparse matrix vector multiplication, sparse matrix addition and sparse matrix matrix multiplication kernels by 4.22×, 6.14× and 6.00× respectively.

***Index Terms*—Sparse Algebra, Vector Computing, Scratchpad Memory**

I. Introduction

Many applications can potentially benefit from vectorized execution for better performance, higher energy efficiency and greater resource utilization [4]. Ultimately, the effectiveness of a vector architecture depends on the quality of the vectorized code [5]. Sparse matrix operations are a clear example of computations difficult to vectorize [3]. Such computations are a key kernel in High Performance Computing (HPC), Artificial Intelligence (AI) and big data workloads. In particular, two such killer-applications are Sparse Matrix Vector multiplication (SpMV) and Sparse Matrix Matrix Multiplication (SpMM). There are two intertwined obstacles against efficient execution of sparse matrix computations on vector architectures: (1) existing sparse matrix representations are not easily vectorizable, and (2) current vector hardware implementations are not optimized for sparse matrix operations.

In this paper, we propose the *Vector Indexed Architecture* (VIA), a vector architecture that aims at accelerating sparse matrix computation. VIA features a smart scratchpad memory specially designed to cope with sparse-dense (SpMV) and sparse-sparse (SpMM) matrix computations.

II. VIA: Key Design Ideas

Sparse matrix kernels present a set of challenges for current Vector Architectures: 1) High usage of inefficient memory indexed instructions and 2) index matching operations. The key idea in VIA is to attach a scratchpad memory (SPM) next to the vector functional units. The VIA SPM features two mapping techniques to tackle both challenges. For *sparse-dense* kernels, VIA performs a direct-mapped access to the SPM. The indexed instructions are executed between the Vector Functional Unit(VFU) and the SPM in VIA, thus reducing memory traffic and releasing memory bandwidth to load the low-locality sparse matrix from main memory more efficiently. For *sparse-sparse* kernels, the SPM in VIA works as a CAM memory. CAM memories are specialized hardware structures that are particularly suitable for search and index matching algorithms. The CAM-based mapping technique, allows VIA to execute index matching operations between two input vectors in a single instruction.

III. VIA: Design Implementation

VIA is composed of two main building blocks: a *Smart Scratch-Pad Memory* (SSPM) and the *Fused Indexed Vector Unit* (FIVU) (see Figure 1). FIVU is the control interface between the Vector Functional Units (VFU) and the SSPM.

A. The Smart Scratchpad Memory

The SSPM is a dedicated high bandwidth structure used to feed the VFU and it can be used in direct-mapped mode, or in CAM-based mode. The SSPM consists of three main building blocks: ❶ the SRAM cells to store the actual data; ❷ the valid bitmap to specify when an entry in the SRAM has been written before; and ❸ the Index tracking logic that provides the CAM-based functionality to SSPM.

SRAM cells (SRAM): stores the values to compute, e.g. for SpMV operations, SSPM stores the vector and for both SpMM and SpMA operations, SSPM stores the sparse row data and indices of only one of the input matrices. In our implementation, SRAM is built using four byte length blocks and each block stores a single value independently on the data length.

Valid bitmap: This structure is used in the direct-mapped mode as a written value indicator for the entries in the SRAM. It consists of a vector of bits, where each bit corresponds to an entry in the SRAM structure and determines whether an entry has been written.

Index tracking logic: This block implements SSPM-CAM functionality over the indexes. The index tracking logic consists of three key components: ❶ The index table, a CAM structure that

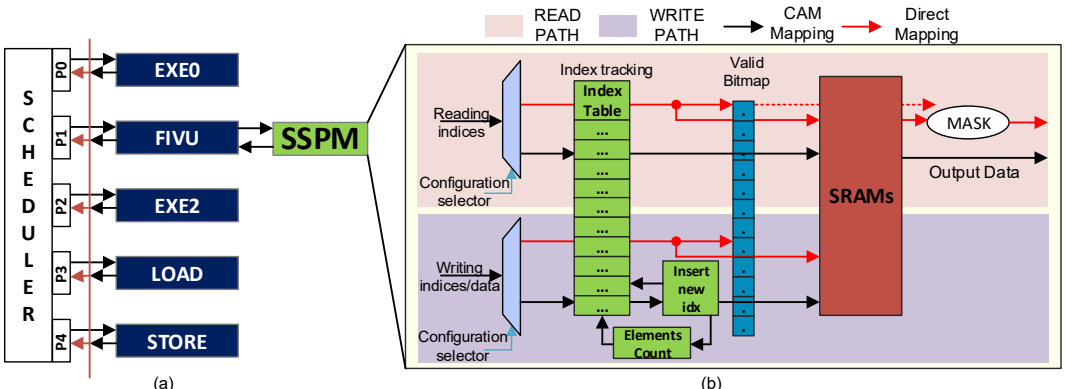

Fig. 1. VIA building blocks: (a) interconnection between FIVU, VIA and the issue logic of an out-of-order core; (b) microarchitecture of SSPM. It consists of the index tracking mechanism (Index table, Insert new Idx and Elements Count), valid bitmap, and the storage system (SRAMs). Read and write paths are depicted separately.

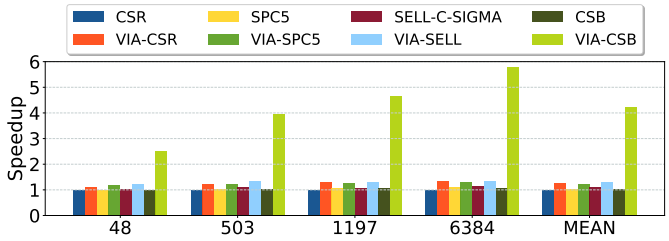

Fig. 2. Speedup for VIA SpMV kernel. Results are normalized to the CSR implementation for every category.

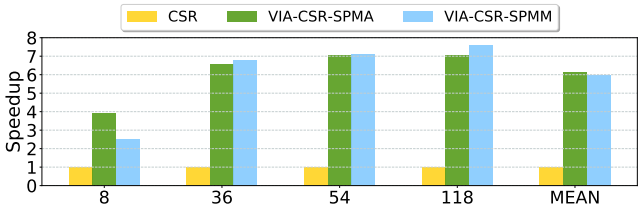

Fig. 3. Speedup for VIA SpMA and VIA SpMM. Both Kernels are normalized to their base CSR implementation.

stores the indices used to write data in the SRAM; ❷ The insertion logic, which inserts new indices and elements in order in the first available position in the index table and the SRAM respectively; and ❸ The element count register, which holds the number of stored indices in the index table.

B. The Fused Indexed Vector Unit

VIA introduces the *Fused Indexed Vector Unit* (FIVU) to operate over data stored in the SSPM. The FIVU works as the interface between the SSPM memory and the processor pipeline and minimally extends a generic Vector Functional Unit (VFU) with new pipeline stages to control operations to the SSPM.

IV. Experimental Setup

We model and evaluate VIA using *Gem5* [1] to simulate an x86 full-system running an Ubuntu 16.04 OS with a 4.9.4 Linux Kernel. We simulate a single core processor using the out-of-order CPU and memory models, extended with the micro-architectural support and performance counters for VIA. We evaluate VIA efficiency using three representative sparse matrix kernels: Sparse Matrix Vector multiplication (SpMV), Sparse Matrix Addition (SpMA) and Sparse Matrix Matrix multiplication (SpMM). As input dataset, we use 1,024 sparse matrices from 56 different application domains of the *University of Florida Sparse Matrix Collection* [2].

V. Evaluation

Figure 2 depicts performance results for VIA-SpMV kernel using different compressed representations on all the input dataset. The most noteworthy results are presented by the CSB (Compress Sparse Block) version. All the evaluated matrices were sorted by the CSB block density and evenly split among 4 categories. The x-Axis at Figure 2 shows the median non-zero values per block among each category. VIA SpMV achieves on average speedup of 4.22× with CSB; and average speedups of 1.25×, 1.24× and 1.31× over the CSR, SPC5 and Sell-C-σ implementations repectively. The CSB format increases the locality of the input and output vectors, thus a chunk of the input vector needs to be placed in SSPM only once to compute with a single block. For the other formats, the indices to map the input or output vectors of two consecutive matrix values can be really sparse, thus the efficiency of VIA is limited to work as an accumulator for the output vector. Nevertheless, even with this limitation, VIA improves performance over the other formats by 1.26× on average. For the best usage case (executing with CSB VIA-SpMV), VIA: (1) reduces the total energy consumption (leakage + dynamic) by a factor of 3.8×. (2) increases the memory bandwidth by 2.5×.

Figure 3 shows the performance of the SpMA (VIA-CSR-SPMA column) and SpMM kernels. In a similar manner to SpMV, results were sorted and evenly split into 4 categories. As we use CSR format in both kernels construction, we used the non-zero elements per row as the criteria to sort the entire input dataset.

On average, VIA achieves 6.14× and 6.0× speedup on the input dataset for SpMA and SpMM respectively. In terms of energy, VIA reduces on average 5.6× and 5.1× of the total energy and increases the memory bandwidth by a factor of 2.1× and 3.2× for SpMA and SpMM respectively. The components of VIA allow to vectorize the index matching computation with a single vector instruction without any extra software comparisons. This capability helps to reduce the memory traffic, reduce the store-load forwarding and to increase the efficiency of the VFU over this kernel.

VI. Author Biography

Julian Pavon was born in Panuco, Mexico, in 1992. He received the B.E degree in Electronic Engineering from the Panuco's Institute of technology, Mexico, in 2015, and the MIRI degree in Research of Informatics from the Universitat Politecnica de Catalunya (UPC) Barcelona, Spain in 2018.

Since March 2018 he has been with the *Computer Architecture for Parallel Paradigms* group, Barcelona Supercomputing Center, where he was a research engineering, and became a PhD student in 2019. His current research topics include vector architectures, Embedded Systems RTL design and RISCV SoC design.

References

[1] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib Bin Altaf, N. Vaish, M. Hill, and D. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, 08 2011.

[2] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[3] E. F. D'Azevedo, M. R. Fahey, and R. T. Mills, "Vectorized sparse matrix multiply for compressed row storage format," in *Computational Science – ICCS 2005*, V. S. Sunderam, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 99–106.

[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[5] N. Satish, C. Kim, J. Chhugani, H. Saito, R. Krishnaiyer, M. Smelyanskiy, M. Girkar, and P. Dubey, "Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications?" in *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA)*, 2012, pp. 440–451. [Online]. Available: http://dl.acm.org.recursos.biblioteca.upc.edu/citation.cfm?id=2337159.2337210