MDPI

Article

Optimizing the performance of SPMV kernel in FPGA guided by the Roofline model

Federico Favaro [1,†]*, Ernesto Dufrechou [2,†], Juan P. Oliver [1,†] and Pablo Ezzatti [2,†]

[1] Instituto de Ingeniería Eléctrica, Facultad de Ingeniería Universidad de la República, Montevideo, Uruguay; {ffavaro,jpo@fing.edu.uy}
[2] Instituto de Computación, Facultad de Ingeniería Universidad de la República, Montevideo, Uruguay; {edufrechou,pezzatti}@fing.edu.uy
* Correspondence: ffavaro@fing.edu.uy
† These authors contributed equally to this work.

Abstract: The widespread adoption of massively parallel processors over the past decade has fundamentally transformed the landscape of High-Performance Computing hardware. This revolution has recently driven the advancement of FPGAs, which are emerging as an attractive alternative to power-hungry many-core devices in a world increasingly concerned with energy consumption. Consequently, numerous recent studies have focused on implementing efficient dense and sparse Numerical Linear Algebra (NLA) kernels on FPGAs. To maximize the efficiency of these kernels, a key aspect is the exploration of analytical tools to comprehend the performance of the developments and guide the optimization process. In this regard, the Roofline Model (RLM) is a well-known graphical tool that facilitates the analysis of computational performance and identifies the primary bottlenecks of a specific software when executed on a particular hardware platform. Our previous efforts advanced in developing efficient implementations of the sparse matrix-vector multiplication (SpMV) for FPGAs, considering both speed and energy consumption. In this work, we propose an extension of the RLM that enables optimizing runtime and energy consumption for NLA kernels based on sparse blocked storage formats on FPGAs. To test the power of this tool, we use it to extend our previous SpMV kernels by leveraging a block sparse storage format that enables more efficient data access.

Keywords: sparse NLA; FPGA; energy consumption; performance modelling

Citation: Favaro, F.; Dufrechou, E.; Oliver, J.P.; Ezzatti, P. Optimizing the performance of SPMV kernel in FPGA guided by the Roofline model. *Micromachines* **2023**, *1*, 0. https://doi.org/

Received:
Revised:
Accepted:
Published:

1. Introduction

The dissemination of massively parallel processors in the last decade irreversibly changed the landscape of High-Performance Computing (HPC) hardware. Recently, this true revolution has propelled the development of Field-Programmable Gate Arrays (FPGAs) as hardware accelerators for HPC applications. Consequently, FPGAs are becoming an appealing alternative to more power-hungry many-core devices in a world increasingly preoccupied with energy consumption [1,2]. One of the key restrictions on the widespread adoption of FPGAs in the software community is their programming model, which is radically different from standard software languages. However, manufacturers have been working to adopt High-Level Synthesis (HLS) languages like System C, C/C++, or OpenCL to facilitate the development and enhance productivity. A remarkable example of this policy is the introduction of SDKs for HLS by the largest FPGA manufacturers, Intel and AMD-Xilinx. Another indicator is the increasing number of research works in HLS [3] and the emergence of multiple HLS tools.

The new developments in the programmability of FPGAs allow these devices to abandon their original niche and be used to solve general-purpose problems. In particular, several recent works on FPGAs focus on the implementation of Numerical Linear Algebra (NLA) kernels [4–6], which are a crucial component of a myriad of applications. However,

most works focus on dense NLA [7–9], as well as other algorithms more suited for FPGAs such as image processing [10,11], machine learning [12], etc. Sparse linear algebra, on the other hand, poses several challenges due to its irregular access patterns and low computational intensity. In this sense, we advanced in developing efficient implementations of the sparse matrix-vector multiplication (SpMv) kernel for FPGAs, both from the viewpoint of speed and energy consumption. In [13], we developed basic SpMv kernels and worked on optimizing their energy consumption. Additionally, we experimentally studied the performance roof of our kernels and included some techniques to overcome the main bottlenecks in [14].

Besides the empirical study, it is also interesting to apply analytical tools to understand our developments' performance and guide these kernels' optimization process. Several research works focus on developing models and frameworks to estimate the performance of FPGA designs [15–17]. In this sense, the Roofline Model (RLM) [18] is a graphical tool that can help analyze the computational performance and the main bottlenecks of a specified software when it runs in a particular hardware platform. Specifically, the RLM is a chart that relates the computational performance (usually measured in floating-point operations per second, FLOPS) with the arithmetic intensity (the ratio of floating-point operations per byte of memory transferred). The chart also shows the memory bandwidth and arithmetic performance bounds of the hardware.

The main contribution of this work is an extension of the RLM that allows the optimization of performance and energy consumption of NLA kernels based on sparse block format storage on FPGAs. The model works on operations per memory access instead of the traditional computational intensity and features a quick mechanism to find the performance and energy efficiency of sparse algebra kernels that use sparse block representation formats. As a minor contribution, we present a SpMv kernel HLS implementation that exploits a block sparse storage format to enable more efficient data access, which we use to put our RLM model to test.

The rest of the paper is structured as follows. In Section 2, we present the SpMv kernel and related works of its implementation in FPGAs as well as the main concepts of the RLM, with a special focus on the main literature about its use with FPGAs. Next, in Section 3, we describe our previous efforts in developing SpMv kernels in FPGA and the new designs for this kernel, and we leverage the RLM to offer a visual procedure to improve, from a runtime and energy consumption perspective, our SpMv method. The experimental evaluation follows this in Section 4. Finally, Section 5 contains the concluding remarks and an outline of future research.

2. The SpMv and the Roofline model

In this section, we present the SpMv kernel and the works related to its implementation in FPGAs. Afterward, we introduce the Roofline model and discuss related works on its utilization in FPGAs.

2.1. The SpMv kernel

Multiplying a sparse matrix by a vector (SpMv) is an essential operation of numerical linear algebra. In many scientific problems, it is one of the most resource-consuming stages. The most typical example is the solution of linear systems of equations using Krylov subspace methods. Such methods imply the repeated use of the SpMv kernel, keeping the same (sparse) matrix but varying the dense vector.

The importance of this kernel has motivated considerable efforts focused on its optimization or performance improvement. These efforts have also accompanied the historical evolution of HPC platforms, and therefore, it is possible to find efficient implementations of the SpMv for the most widespread hardware platforms. However, as this evolution is constant, new implementations and parallel algorithms for the SpMv are still being proposed. In Section 2.2, we present related works about efficient implementations of SpMv in FPGAs.

Algorithm 1 summarizes the serial version of the SPMV, where the sparse matrix A is stored in Compressed Sparse Row (CSR) format [19]. The non-zero elements are stored in vector val, the column index of each element in the matrix A is stored in a vector col_idx, and row_ptr stores the index of the first element for each row in vector val. The non-zero elements within each row are arranged by column index.

Algorithm 1 Serially computed sparse matrix-vector multiplication (SPMV) with the sparse matrix A stored in CSR format.

Input: row_ptr, col_idx, val, x
Output: y

```
y = 0
for i = 0 to n − 1 do
    for j = row_ptr[i] to row_ptr[i + 1] − 1 do
        y[i] = y[i] + val[j] · x[col_idx[j]]
    end for
end for
```

The main approach to parallelizing this operation using the CSR format is to exploit the absence of data dependencies between the computations associated with each row. However, this approach presents severe load imbalances (depending on the sparsity pattern) and suffers from indirect data access to the dense vector x.

2.2. SPMV *in FPGA*

In [20], the authors present an architecture for the SPMV that can process non-zero elements from multiple rows in parallel. They introduce a variation of the CSR matrix representation format called CSR Interleaved to achieve this. Furthermore, they eliminate the necessity of maintaining separate copies of the vector x for each multiplier by introducing the concept of a Banked Vector Buffer (BVB). In this approach, the vector x is divided into 32 distinct memory banks, and a crossbar is utilized to direct non-zero elements to their respective vector banks.

In [21], the authors develop a streaming dataflow architecture for the SPMV operation and test it in a Xilinx ZynqMP FPGA standalone board. Their proposed solution features a deep pipeline that continuously consumes input data without stalling. To add parallelism, they simultaneously process multiple rows in separate compute pipelines, each with its own I/O ports. This requires merging row and column indices within the same input port due to the limited number of I/O ports available on the device.

The authors in [22] analyze the performance and energy efficiency of SPMV and other kernels in two FPGA platforms, one with DDR memory and the other with HBM, and compare the results with multi-core CPUs and a GPU. They include the Empirical Roofline Model (using the Empirical Roofline Toolkit extended for FPGAs) to enhance the platform characterization and cross-platform comparison. They find that FPGAs are well behind GPUs in absolute terms for computing and memory-intensive kernels but require far less power and are competitive in energy efficiency.

Yixiao Du et al. [23] developed HiSparse, a high-performance kernel for HBM-equipped FPGAs. HiSparse uses a custom format for storing sparse matrices in HBM, allowing vectorized streaming access to each HBM channel and concurrent access to multiple channels. This helps to fully utilize the available bandwidth of HBM for loading the sparse matrix. In addition, HiSparse features a scalable on-chip buffer design that incorporates vector replication and banking to support a large number of parallel processing engines (PEs). This helps to maximize data reuse in accessing the input vector. HiSparse can support arbitrarily large matrices through the use of matrix partitioning along both rows and columns.

Xilinx offers its own set of open-source HLS libraries known as Vitis libraries, which include one for the BLAS specification and another for sparse operations [24]. These libraries provide optimized blocks for sparse data movement and calculations, as well as an SPMV kernel that leverages these blocks. The SPMV kernel provided by the library is

specifically optimized for Xilinx's Alveo architectures, efficiently utilizing HBM memory and a matrix partitioning scheme. To our knowledge, this represents the most high-performing SpMv implementation on Alveo platforms. It's worth noting that the objective of our work is not to deliver the best performance implementation but rather to present it as a case study to demonstrate the application of the proposed RLM techniques on FPGAs.

2.3. Roofline Model (RLM)

The Roofline Model (RLM) is a graphical tool that provides performance estimates for a computational kernel running on a specific hardware platform. It is particularly valuable for identifying performance bottlenecks and offering optimization guidance, helping programmers and architecture designers achieve maximum computational efficiency.

At its core, the Roofline Model combines three key concepts within a single plot: computational peak performance (CP) measured in GFLOP/s, which represents the maximum number of floating-point arithmetic operations that can be executed per second, memory bandwidth (BW) measured in GB/s, that quantifies the maximum rate at which data can be transferred between off-chip memory and computational units, and computational intensity (CI), which measures the extent of data reuse between off-chip memory and computational units. Computational intensity is often calculated as the ratio of useful operations performed to the amount of memory transferred to and from off-chip memory in a given computational kernel.

Figure 1 summarizes the fundamental concepts of the model, displaying the performance ceiling in GFLOPS and the bandwidth ceiling in GB/s. The point of intersection between peak bandwidth and peak performance is referred to as the ridge point. Applications located to the left of this ridge point are considered memory-bound, indicating that their performance is primarily constrained by memory bandwidth. Conversely, applications to the right of the ridge point are classified as compute-bound, implying that the computational resources of the hardware platform predominantly limit their performance. In Figure 1, Algorithm 1 has a low CI, which causes it to be limited by bandwidth. As the CI increases, Algorithm 2 becomes compute-bound.

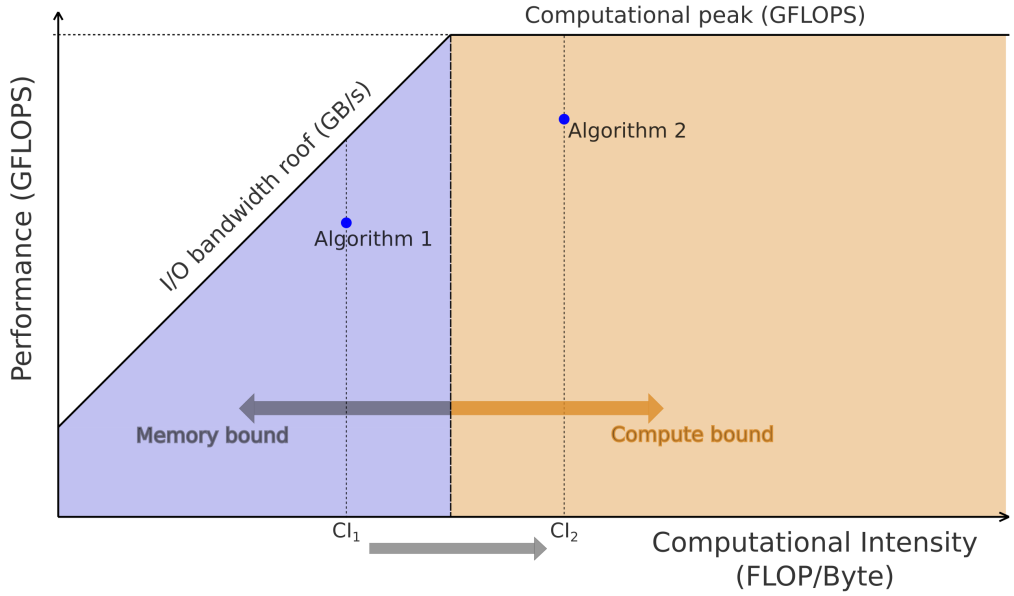


Figure 1. Roofline Model graph.

When employing the Roofline Model with FPGAs, some considerations should be considered. Firstly, it's important to note that FPGAs lack a fixed architecture; their architecture is determined by the algorithm in use. As a result, the performance ceiling depends on the specific algorithm employed. While it's possible to establish an absolute

performance limit based on the hardware resources available for arithmetic operations, this value may deviate significantly from reality depending on the chosen algorithm. Additionally, it's crucial to recognize that some resources are used for routing, and the clock frequency is also influenced by the design. On a positive note, in FPGAs, the designer has complete control over the management of internal (fast) memories, which allows them to precisely determine the number of memory accesses at all levels. This makes it possible to accurately calculate computational intensity, unlike in CPU architectures, where obtaining this value can be challenging because of the hard-to-predict cache-access patterns.

2.4. RLM in FPGAs

Although the Roofline model is not completely widespread in the FPGA community, some interesting published works can be found. Da Silva et al. [25] propose using RLM for each algorithm optimization in FPGA with High-Level Synthesis tools. The authors employ different metrics, expressing the computational intensity as ByteOperations over ByteMemoryAccess, and the performance as ByteOperations per second.

M. Pahlavan [26] adapts the RLM for FPGAs in his Master Thesis. Pahlavan proposes the FPGA-Roofline, focusing on defining the bandwidth link as the main bottleneck and representing the available throughput (operations per second) per operational intensity.

E. Calore [27] introduces the FPGA Empirical Roofline (FER), which serves as a benchmarking instrument for empirically assessing computing performance and external memory bandwidth of FPGAs in HPC applications. Their approach allows cross-architectural comparisons and provides an estimation of the performance potential for kernels developed using HLS tools according to their arithmetic intensity.

I. Panagou [28] propose an application-centric Roofline model that takes into account both resource limitations and latency constraints, attaining a more precise Roofline ceiling. They expand this model to support multiple kernels implemented within the FPGA fabric and on the CPU in MPSoC platforms. Additionally, considering that the arithmetic intensity may vary across various inputs, they propose an input-driven model that enables better optimization strategies.

Recently, D. Diakite et al. [29] leveraged the RLM to optimize a tomography application in OpenCL in an FPGA. Authors define the performance roof as the number of resources consumed and the effective operation frequency, and the graph employs the same axis as M. Pahlavan's proposal.

T. Nguyen et al. [30] deeply explore the RLM use in the FPGA context. Additionally, the effort addresses the energy consumption problem and the SpMV kernel. M. Siracusa et al. [31] extend this work by defining a comprehensive methodology to optimize FPGA designs based on the RLM.

3. Implemented kernel and RLM

In this section, we first present the SpMV kernel implementation. Next, we explain how we use the roofline model to improve the kernel implementation, and finally, we incorporate the analysis of energy consumption into the model.

3.1. Implemented Kernel

In a previous effort, we developed an FPGA kernel to compute SpMV operation in CSR format. The kernel proceeds row-wise and, to alleviate the indirect accesses to vector x, the vector is cached entirely in the FPGA's internal memory. We opted for a coarse-grained parallelism approach to optimize the kernel, instantiating it multiple times (Compute Unit Replication). However, as this also implies the replication of the x cache, the number of CUs is severely limited by the FPGA resources.

In a posterior effort, we explored fine-grained parallelism (i.e., vectorizing the kernel's datapath), requiring a completely different kernel architecture and sparse representation formats.

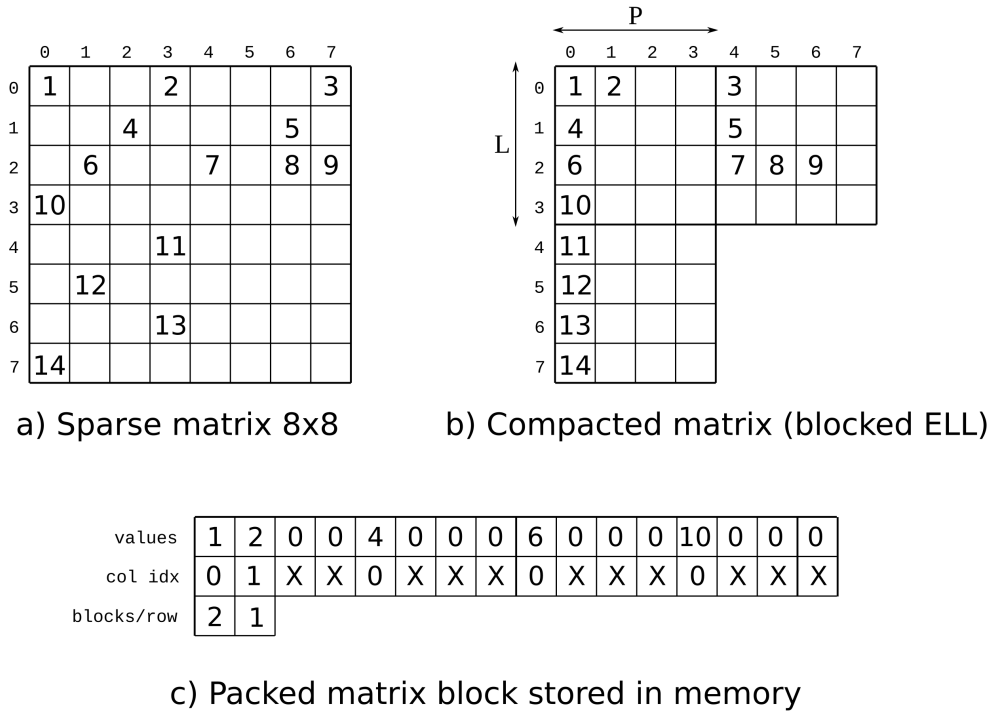


Figure 2. Example of the blocked ELLPACK sparse format with a 8×8 matrix, $L = 4$ and $P = 4$.

One of the main obstacles in implementing the SpMv in FPGA is that the floating-point addition has considerable latency. Moreover, the product and accumulation of the partial results (line 4 in Algorithm 1) presents a data dependency between loop iterations. This dependency prevents the operation from producing one result per clock cycle (i.e., pipeline parallelism). A method to "hide" this data dependency is to interleave the processing of L rows simultaneously, where L is the latency of the floating-point addition.

For this strategy to be efficient, data must be interleaved before it is read by the FPGA, allowing contiguous memory access. To facilitate this, we opted for the sliced ELL format, which packs the non-zero elements (and their indices), creating dense blocks. We divide the matrix into partitions of L rows and then apply ELL to each partition. The longest row determines the width of each block, padding the rest of the rows with zeros to match its length. Then the blocks are stored in memory so that the L rows are interleaved (blocks are transposed). We refer to the implemented kernel architecture that uses this format as *Scalar*.

The issue with this format is that when trying to vectorize the kernel, non-contiguous elements of the vector x are simultaneously accessed. The previous format is extended to address this problem so that the blocks have a fixed width P (with ideas similar to the SELL-P sparse storage format [32]). This way, the elements of x for each block do not have a distance greater than P. Then we can partition the x cache in the FPGA to allow P simultaneous accesses. We show an example of the sparse format in Figure 2. In addition to the data and column indices, the format requires a vector indicating the number of dense blocks per row of blocks. We refer to this format as Blocked-ELL and to the implemented kernel architecture that uses this format as *Block*.

A diagram of the resulting kernel, which was implemented using Xilinx's HLS language, is shown in Figure 3. The architecture consists of two parts. The first, on the left, is in charge of reading the data from memory and arranging it before it is passed to the second part, which performs the computations. All the blocks are instantiated in parallel, work simultaneously (i.e., dataflow operation), and connect by FIFO channels. The datapath has a width of P elements (floats for values and x, and int for indices). The computation block has two available reducers and switches between them to avoid delays.

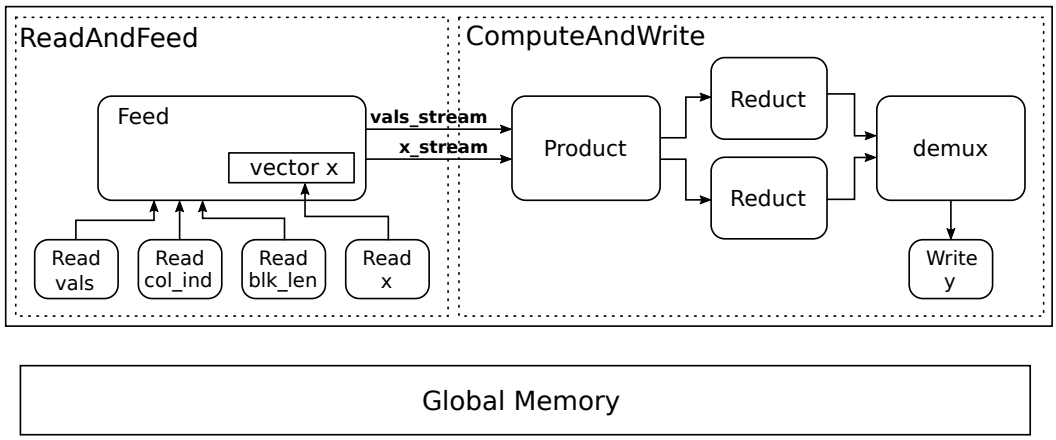


Figure 3. Block diagram of the implemented SpMv kernel with fined-grained parallelism.

3.2. The RLM as a guide to optimizing the performance of blockwise SpMv

The reviewed works in section 2.4 provide detailed information about employing the RLM in optimizing the SpMv for implementations where the operation count and data transfers can be directly derived from the number of non-zeros of the sparse matrix. However, these models are inaccurate for sparse formats such as ELLPACK, where a certain amount of zero-padding is allowed to improve memory performance. Therefore, we propose an adaptation of the RLM that considers the zero-padding to study the bottlenecks in our sparse matrix kernels. In other words, we offer a methodology to leverage the RLM to optimize block-sparse codes in FPGAs, where the key is the procedure to increase the computational intensity, when the kernel is in the memory bound region (blue zone in Figure 1). The advantage of the RLM is that it provides straightforward visual verification of the procedure. Some related ideas were previously presented for dense algebra methods running in GPUs [33].

We define the model's vertical axis as performance in GFLOPs and the horizontal axis as CIA (Computational Intensity per Access), representing floating-point operations per memory access. For sparse blocked formats, this metric is related to the density of the blocks. With this definition of CIA, the bandwidth roof is expressed in giga memory accesses per second. Consequently, an increase in the operation count due to a rising amount of data transferred in each memory access impacts the intensity but not the bandwidth.

Sparse codes that employ zero-padded blocks reach two different intensities: the theoretical intensity, defined by the block's dimension, and the useful intensity, which only considers the number of non-zero coefficients in each block. We employ the second one, computed using the average of non-zero coefficients per block, since counting floating-point operations performed with zeros artificially increases computational intensity and performance. Therefore, the CIA reached by the SpMv in our model depends on the block dimension and the sparse matrix pattern.

In summary, a kernel's position within our RLM extension depends not only on the computing hardware platform (the FPGA) and the (block) computer solver, as it is typically the case in RLM, but also on the non-zero coefficient pattern of the matrix.

3.3. Extending the proposal to address energy consumption

To evaluate the energy consumption, we include more information in our extension of the RLM. We draw a line parallel to the X-axis corresponding to the peak of performance of the *Scalar* method. Then, we draw another parallel line corresponding to the performance that matches the energy consumption of the *Scalar* method with the power of the *Block* method. In other words, if P_1 is the average power of the *Scalar* method, P_2 is the average power of the *Block* method, and *flop* is the number of required floating-point operations, then the energy spent in the two lines is $E = P_1 t_1 = P_2 t_2$. Therefore, $t_2 = (P_1/P_2)t_1$, and the second line is drawn at $perf_2 = flop/t_2 = (P_2/P_1)(flop/t_1) = (P_2/P_1)perf_1$. This defines

two zones, the first where the *Block* algorithm decreases the runtime but increases the energy consumption, and the second, where both metrics are improved. Furthermore, the graph shows the minimum CIA for improving energy consumption. Figure 4 summarizes the idea, where the first zone is red and the second is green.

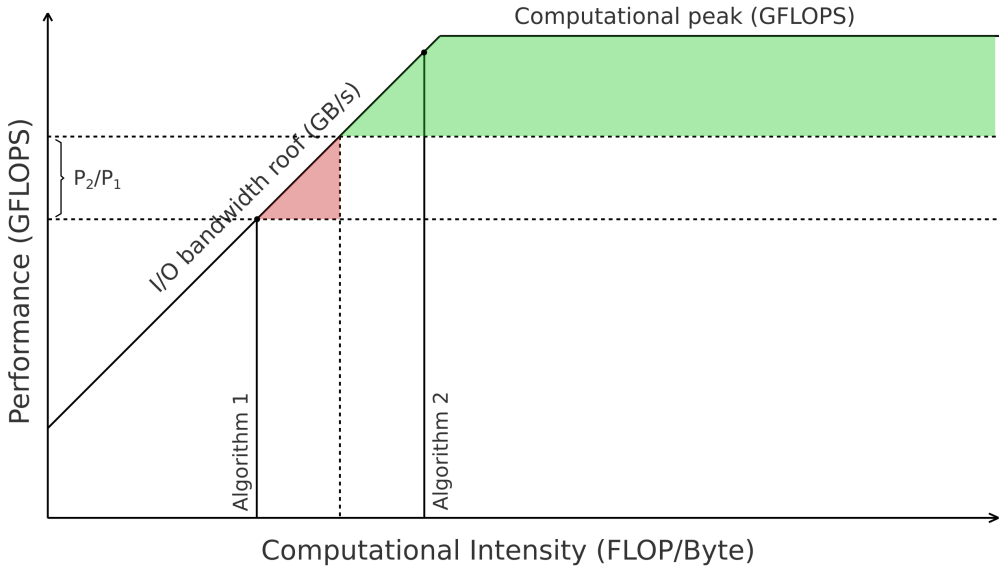


Figure 4. Extension of RLM for runtime and energy consumption optimization.

4. Experimental Evaluation

In this section, we present the experiments carried out to evaluate the performance of our kernels. First, we describe our computing platforms and the equipment used to evaluate runtime and power consumption. Next, we present the test cases and the numerical results, accompanied by the corresponding analysis. Last, we analyze the results using the extended RLM variant considering energy consumption.

4.1. Experimental Setup

We used the Alveo U50 board for the experimental evaluation, an acceleration platform for data center applications based on a Xilinx FPGA with UltraScale+ architecture. This board includes 8GB of built-in High Bandwidth Memory DRAM and QSFP28 connections for 100GbE applications. Its maximum power consumption is 75W, which, added to the high computing capacities and memory bandwidth, achieves excellent energy efficiency. It has 872K logic elements, 1,743K registers, 28MB of internal RAM, and 5,952 DSP blocks. We compiled the cores for the Alveo platform using Xilinx Vitis 2020.2. For comparison purposes, we have included a traditional CPU system with two Intel Xeon Silver 4208 processors with eight cores each and 80GB of RAM.

We measured the power consumption as follows:

- In the Alveo U50, we use the board's internal sensors that provide current, voltage, and temperature readings while the kernel is running. The driver Xilinx Runtime (XRT) sends these values to the host.
- For the Intel processor, we measure CPU and memory consumption using RAPL (which estimates the dissipated power based on performance counters and a device power model).

We obtain the execution times through the operating system libraries in the case of the Intel processor and OpenCL profiling functions in the FPGA. The runtime measurements are the average of multiple iterations of the cores. In the case of the power measurements,

the measured power is the average of the readings collected within 3 minutes of execution, with a warm-up time of 2 minutes.

4.2. Test cases

We validate the proposal using matrices from the SuiteSparse Matrix Collection (formerly known as UF Sparse Matrix Collection). We choose matrices with diverse characteristics and non-zero patterns, with dimensions (n) from 17,000 to 40,000 and a number of non-zeros (nnz) between 14,765 and 16,171,169. Table 1 shows the main characteristics of the matrices.

Table 1. Evaluated matrices, with number of rows (n) and non-zero elements (nnz). The last column is the average density of the blocks resulting from the blocked ELL format for block size 16×16.

Matrix	n	nnz	avg density of blocks (%)
bcsS37	25503	14765	4
bcsS35	30237	18211	4
qpband	20000	30000	5
ex9	3363	51417	21
body4	17546	69742	6
body6	19366	77057	6
ted_B	10605	77592	20
ted_B_un	10605	77592	20
nasa2910	2910	88603	24
s3rmt3m3	5357	106526	27
s2rmq4m1	5489	143300	34
chipC0	20082	281150	1
cbuckle	13681	345098	31
olafu	16146	515651	33
gyro_k	17361	519260	10
GoodW40	17922	561677	18
bcsstk36	23052	583096	27
msc23052	23052	588933	13
msc10848	10848	620313	18
raefsky4	19779	674195	27
TS162	20374	812749	31
nd3k	9000	1644345	43
thread	29736	2249892	24
TS300	28338	2943887	51
nd6k	18000	3457658	42
TS2052	25626	6761100	70
TS2383	38120	16171169	71

4.3. Experimental results

This section summarizes the experimental results for the *Scalar* and *Block* methods. For the *Block* kernel, we used block size 16×16. For comparison purposes, we include the results for the CPU running MKL implementation of SpMv. Table 2 includes the runtime and energy consumption of the three variants. The results show that the new *Block* algorithm, which uses fine-grained parallelism, significantly reduces the execution times of the *Scalar* version in most cases. Another noteworthy aspect is that the reductions are more important as nnz increases, reaching a maximum of $7.5\times$. For some of the smallest matrices, the *Block* method decreases the performance. This is because the sparsity pattern and the number of non-zeros lead to excessive padding. The CPU version outperforms the FPGA methods in all cases.

Table 2. Runtime and energy results for CPU and FPGA implementations.

Matrix	t_{CPU} MKL (ms)	t_{FPGA} *Scalar* (ms)	t_{FPGA} *Block* (ms)	E_{CPU} MKL (mJ)	E_{FPGA} *Scalar* (mJ)	E_{FPGA} *Block* (mJ)
bcsS37	0.012	0.280	0.324	0.6	5.2	6.6
bcsS35	0.012	0.331	0.366	0.6	6.2	7.5
qpband	0.017	0.258	0.356	0.8	4.8	7.3
ex9	0.015	0.409	0.195	0.7	7.6	4.2
body4	0.024	0.414	0.698	1.1	7.7	14.9
body6	0.025	0.434	0.791	1.2	8.1	16.9
ted_B	0.031	0.625	0.285	1.5	11.7	6.1
ted_B_un	0.037	0.686	0.297	1.7	12.8	6.3
nasa2910	0.022	0.890	0.243	1.0	16.6	5.2
s3rmt3m3	0.026	0.595	0.259	1.2	11.1	5.5
s2rmq4m1	0.028	0.686	0.273	1.2	12.8	5.8
chipC0	0.067	1.576	7.772	3.1	29.4	167.1
cbuckle	0.073	1.775	0.738	3.2	33.1	15.7
olafu	0.117	2.527	1.008	5.3	47.1	21.5
gyro_k	0.097	2.754	1.837	4.7	51.4	39.2
GoodW40	0.104	4.064	1.078	5.0	75.9	23.1
bcsstk36	0.130	3.014	1.240	6.1	56.2	26.5
msc23052	0.183	3.052	2.097	8.5	56.9	44.7
msc10848	0.181	3.546	1.703	8.2	66.1	36.3
raefsky4	0.125	3.097	1.369	6.3	57.7	29.2
TS162	0.216	3.140	0.991	10.4	58.6	21.0
nd3k	0.322	6.755	1.841	15.9	126.0	39.3
thread	0.448	11.120	3.102	23.5	207.9	67.5
TS300	0.986	10.290	2.024	49.3	192.3	43.7
nd6k	0.761	13.962	2.839	40.1	261.4	62.3
TS2052	2.839	23.093	3.284	139.4	431.2	72.1
TS2383	6.411	54.543	7.253	323.7	1019.5	159.5

Regarding the energy consumption, due to the reduced power consumption of the FPGA, for the cases where the execution time between both platforms is comparable (large matrices) the FPGA is more energy efficient. For the two largest matrices the FPGA is 2× more efficient than the CPU in this aspect.

The execution time on the FPGA is linear with NNZ after padding. This does not hold true for the CPU. First, it can be observed that when NNZ exceeds the cache level, there is a relative increase in execution time. On the other hand, there are some cases in which matrices with a higher NNZ run faster than those with fewer nonzeros. Examples of this are: qpband vs ex9, ted_B vs nasa2910, and TS300 vs nd6k. It is not easy to determine a cause for this because we do not have access to the kernel that executes MKL. However, it can be observed that in these cases, the matrices that execute faster, even though they have a higher NNZ, have smaller dimensions (rows and columns). On the FPGA side, if we compare nd6k with TS300, we can see that the first has more padding than the second, directly affecting the CIA and the achieved GFLOPs. This and the above explain why the TS300 matrix is more energy-efficient on the FPGA, whereas the nd6k matrix, despite having more nonzeros, is more efficient on the CPU.

We selected three matrices as an example for the Roofline analysis between the two FPGA variants: chipC0, gyro_k and TS2383. These cases represent three distinct scenarios. In the first one, the *Block* version drastically reduces the performance, for the second one, both methods yield similar runtimes, and for the third one, the *Block* method greatly increases performance. In Figure 5 we show the RLM graph for the *Scalar* (blue) and *Block* (red) results of the three matrices. We include, in green, the maximum attainable

performance of the *Block* kernel (*BlockMax*). This represents the attainable performance of the *Block* method when all the matrix elements are taken into account, including the added zeros.

We obtain the peak performance roofs for each method as the maximum number of FLOPs that the kernel architecture can attain, considering the number of parallel floating-point operations and the clock frequency. The kernels approach the performance roof when the average zero-padding per block tends to zero. For the memory access roof (measured in memory access per second), we first measure the maximum bandwidth (in GB/s) of one HBM channel with an interface of 512 bits using a kernel specifically designed for that purpose. Then, we convert this measure into accesses per second by dividing by the number of bytes of one access. Finally, we adjust the value to consider the number of channels used in the kernel.

This example illustrates the benefits of our proposal. By calculating the CIA for a specific matrix and block size, we can evaluate the effectiveness of the *Block* version without executing the kernel. To achieve this, we draw a vertical line on the computed CIA and determine the attainable performance by intersecting this line with the corresponding roof of the RLM. This approach offers an analytical tool for predicting the runtime and energy consumption behavior of a kernel for each matrix, eliminating the necessity for the time-consuming process of compiling and running the design on an FPGA.

In the same line, Figure 6 summarizes the behavior of the SpMV kernel for various matrices using Blocked-ELL with block sizes of 8×8 and 16×16. In these cases, the utilization of 16×16 blocks outperforms their 8×8 counterpart, as they attain higher values of CIA in the memory-bound zone of the RLM. Furthermore, with the assistance of the RLM, it becomes straightforward to identify which matrices benefit from blocked kernels and which are better suited for the scalar version. It can be observed that the points situated below the *peak Perf Scalar* line will not experience a performance improvement compared to the *Scalar* version, resulting in poorer energy performance (see section 3.3). Conversely, points positioned above the P2/P1 line demonstrate a noticeable performance improvement, leading to a reduction in energy consumption.

5. Concluding remarks

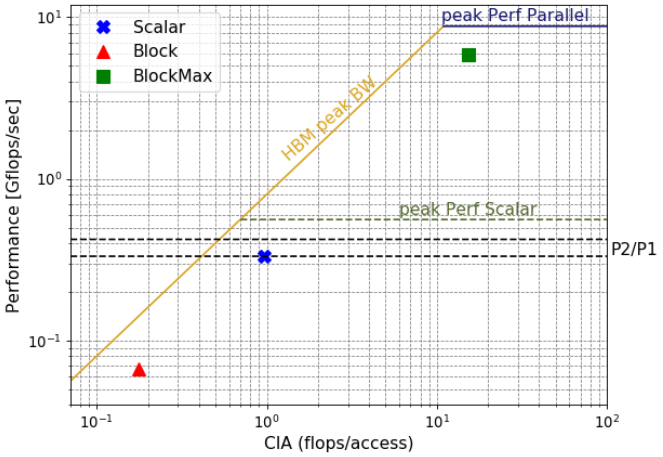
In this paper, we have proposed an extension of the Roofline model for FPGAs intended to optimize sparse linear algebra kernels based on blocked representation formats, both from the runtime and energy consumption perspectives. In particular, we implemented a SpMV kernel that leverages a block sparse storage format to attain concurrency in the data access and parallelism in the computations. The experimental evaluation of our methods, performed on a cutting-edge FGPA, confirms the benefits of this kind of block format for sparse algebra. In the same line, the proposed use of the RLM allows the selection of the best option (between the *Scalar* algorithm and the *Block* algorithm with different block dimensions) to improve the runtime and energy consumption effortlessly.

We plan to address several lines in future work. As the first step, we plan to include specific optimizations on our kernel to improve the performance. Secondly, we expect to extend our ideas to other sparse NLA kernels, such as the SpMM, SpTrSv, and SpGEMM, evaluating the usefulness of our RLM extension. Finally, sparse matrix reordering techniques can be applied to understand the impact of the computational intensity on our kernels' performance.

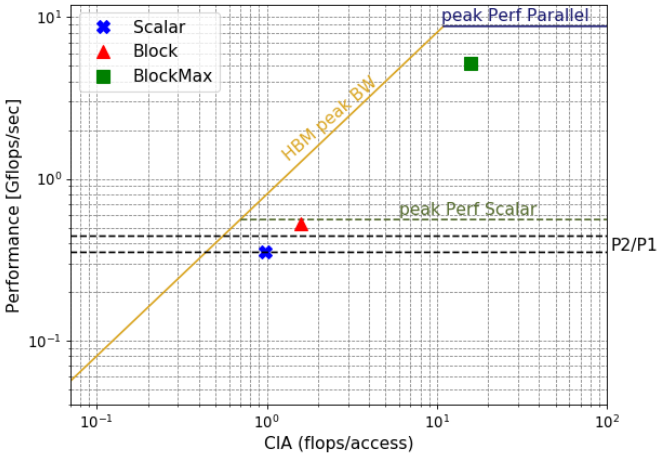
Author Contributions: Conceptualization, Pablo Ezzatti; Software, Federico Favaro; Supervision, Pablo Ezzatti; Validation, Federico Favaro; Writing – original draft, Federico Favaro and Pablo Ezzatti; Writing – review & editing, Ernesto Dufrechou, Juan Oliver and Pablo Ezzatti.

Institutional Review Board Statement: Not applicable.

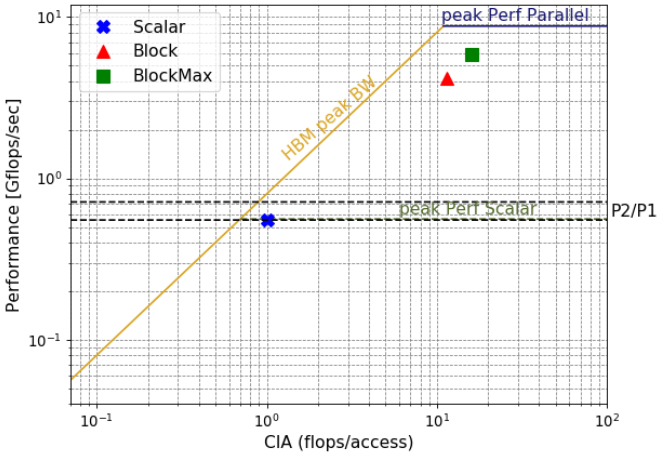
Informed Consent Statement: Not applicable.



(a)



(b)



(c)

Figure 5. Roofline model of three different matrices: (a) chipC0, (b) gyro_k, and (c) TS2383. P2 represents the power consumption of the *Block* version and P1 the power of the *Scalar* version.

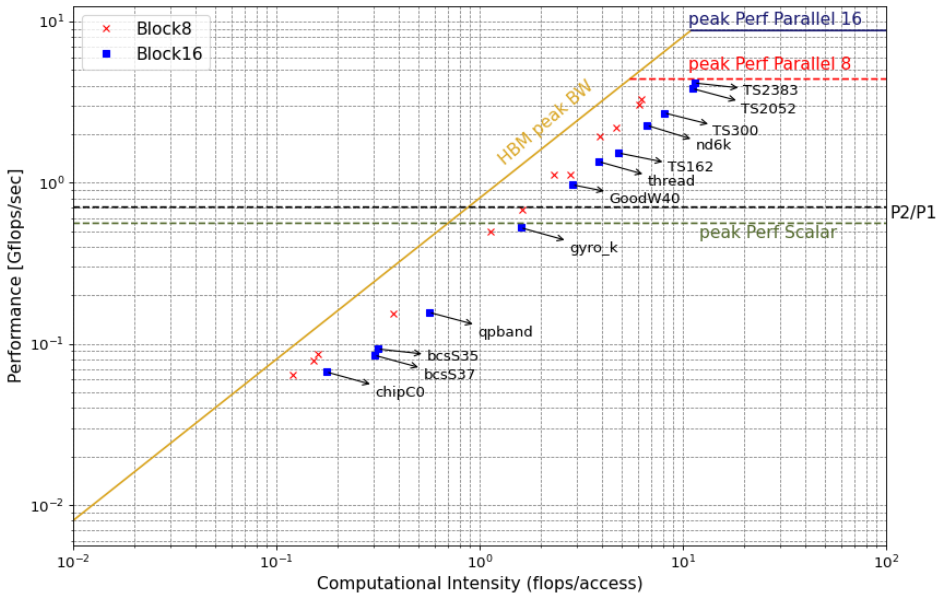


Figure 6. Roofline model for several matrices using block size of 8 (red cross) and 16 (blue square).

Acknowledgments: The researchers were supported by Universidad de la República and the PEDECIBA. We acknowledge the ANII-MPG Independent Research Groups: "Efficient Heterogenous Computing" with the CSC group."

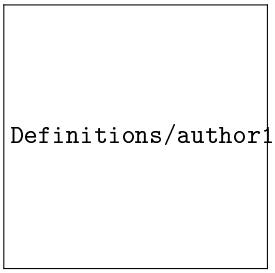
Conflicts of Interest: The authors declare no conflict of interest.

References

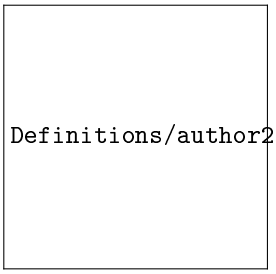
1. Kaxiras, S.; Martonosi, M. *Computer Architecture Techniques for Power-Efficiency*; Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2008. https://doi.org/10.2200/S00119ED1V01Y200805CAC004.
2. Ezzatti, P.; Quintana-Ortí, E.S.; Remón, A.; Saak, J. Power-aware computing. *Concurrency and Computation: Practice and Experience* **2019**, *31*, e5034. e5034 cpe.5034, https://doi.org/10.1002/cpe.5034.
3. Cong, J.; Lau, J.; Liu, G.; Neuendorffer, S.; Pan, P.; Vissers, K.; Zhang, Z. FPGA HLS Today: Successes, Challenges, and Opportunities. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*.
4. De Matteis, T.; de Fine Licht, J.; Hoefler, T. FBLAS: Streaming Linear Algebra on FPGA. In Proceedings of the Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, 2020, SC '20.
5. Arora, A.; Anand, T.; Borda, A.; Sehgal, R.; Hanindhito, B.; Kulkarni, J.; John, L.K. CoMeFa: Compute-in-Memory Blocks for FPGAs. *CoRR* **2022**, *abs/2203.12521*, [2203.12521]. https://doi.org/10.48550/arXiv.2203.12521.
6. Que, Z.; Nakahara, H.; Nurvitadhi, E.; Boutros, A.; Fan, H.; Zeng, C.; Meng, J.; Tsoi, K.H.; Niu, X.; Luk, W. Recurrent Neural Networks With Column-Wise Matrix-Vector Multiplication on FPGAs. *IEEE Trans. Very Large Scale Integr. Syst.* **2022**, *30*, 227–237. https://doi.org/10.1109/TVLSI.2021.3135353.
7. Kestur, S.; Davis, J.D.; Williams, O. BLAS Comparison on FPGA, CPU and GPU. In Proceedings of the 2010 IEEE Computer Society Annual Symposium on VLSI, July 2010, pp. 288–293.
8. Giefers, H.; Polig, R.; Hagleitner, C. Analyzing the energy-efficiency of dense linear algebra kernels by power-profiling a hybrid CPU/FPGA system. In Proceedings of the 2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, June 2014, pp. 92–99.
9. de Fine Licht, J.; Kwasniewski, G.; Hoefler, T. Flexible Communication Avoiding Matrix Multiplication on FPGA with High-Level Synthesis. In Proceedings of the Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, New York, NY, USA, 2020; FPGA '20, p. 244–254. https://doi.org/10.1145/3373087.3375296.
10. Hill, K.; Craciun, S.; George, A.; Lam, H. Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA. In Proceedings of the 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP). IEEE, 2015, pp. 189–193.
11. Díaz, L.; Moreira, R.; Favaro, F.; Dufrechou, E.; Oliver, J.P. Energy Measurement Laboratory for Heterogeneous Hardware Evaluation. In Proceedings of the 2021 IEEE URUCON, 2021, pp. 268–272. https://doi.org/10.1109/URUCON53396.2021.9647059.

12. Mittal, S. A survey of FPGA-based accelerators for convolutional neural networks. *Neural computing and applications* **2020**, *32*, 1109–1139.
13. Favaro, F.; Dufrechou, E.; Ezzatti, P.; Oliver, J.P. Exploring FPGA Optimizations to Compute Sparse Numerical Linear Algebra Kernels. In Proceedings of the ARC. Springer, 2020, Vol. 12083, *Lecture Notes in Computer Science*, pp. 258–268. https://doi.org/10.1007/978-3-030-44534-8_20.
14. Favaro, F.; Oliver, J.P.; Ezzatti, P. Unleashing the computational power of FPGAs to efficiently perform SPMV operation. In Proceedings of the 40th International Conference of the Chilean Computer Science Society, SCCC 2021, Chile, 2021. IEEE, 2021, pp. 1–8. https://doi.org/10.1109/SCCC54552.2021.9650418.
15. Wang, Z.; He, B.; Zhang, W.; Jiang, S. A performance analysis framework for optimizing OpenCL applications on FPGAs. In Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2016, pp. 114–125. https://doi.org/10.1109/HPCA.2016.7446058.
16. Farahmand, F.; Ferozpuri, A.; Diehl, W.; Gaj, K. Minerva: Automated hardware optimization tool. In Proceedings of the 2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2017, pp. 1–8. https://doi.org/10.1109/RECONFIG.2017.8279804.
17. Makrani, H.M.; Farahmand, F.; Sayadi, H.; Bondi, S.; Dinakarrao, S.P.; Homayoun, H.; Rafatirad, S. Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design. In Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Los Alamitos, CA, USA, sep 2019; pp. 397–403. https://doi.org/10.1109/FPL.2019.00069.
18. Williams, S.; Waterman, A.; Patterson, D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* **2009**, *52*, 65–76. https://doi.org/10.1145/1498765.1498785.
19. Golub, G.H.; van Loan, C.F. *Matrix Computations*, fourth ed.; JHU Press, 2013.
20. Fowers, J.; Ovtcharov, K.; Strauss, K.; Chung, E.S.; Stitt, G. A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication. In Proceedings of the 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, 2014, pp. 36–43. https://doi.org/10.1109/FCCM.2014.23.
21. Hosseinabady, M.; Nunez-Yanez, J.L. A Streaming Dataflow Engine for Sparse Matrix-Vector Multiplication using High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2019**, *39*, 1272–1285.
22. Nguyen, T.; MacLean, C.; Siracusa, M.; Doerfler, D.; Wright, N.J.; Williams, S. FPGA-based HPC accelerators: An evaluation on performance and energy efficiency. *Concurrency and Computation: Practice and Experience* **2022**, *34*, e6570, [https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6570]. https://doi.org/https://doi.org/10.1002/cpe.6570.
23. Du, Y.; Hu, Y.; Zhou, Z.; Zhang, Z. High-Performance Sparse Linear Algebra on HBM-Equipped FPGAs Using HLS: A Case Study on SpMV. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)* **2022**.
24. Xilinx Inc.. Vitis SPARSE Library. https://xilinx.github.io/Vitis_Libraries/sparse/2020.2/index.html, 2020.
25. da Silva, B.; Braeken, A.; D'Hollander, E.H.; Touhafi, A. Performance Modeling for FPGAs: Extending the Roofline Model with High-Level Synthesis Tools. *Int. J. Reconfigurable Comput.* **2013**, *2013*, 428078:1–428078:10. https://doi.org/10.1155/2013/428078.
26. Yali, M.P. FPGA-Roofline: An Insightful Model for FPGA-based Hardware Accelerators in Modern Embedded Systems. Master's thesis, Faculty of the Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 2014.
27. Calore, E.; Schifano, S.F. Performance assessment of FPGAs as HPC accelerators using the FPGA Empirical Roofline. In Proceedings of the 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), 2021, pp. 83–90. https://doi.org/10.1109/FPL53798.2021.00022.
28. Panagou, I.; Gkeka, M.; Patras, A.; Lalis, S.; Antonopoulos, C.D.; Bellas, N. FPGA Roofline modeling and its Application to Visual SLAM. In Proceedings of the 2022 32nd International Conference on Field-Programmable Logic and Applications (FPL), 2022, pp. 130–135. https://doi.org/10.1109/FPL57034.2022.00030.
29. Diakite, D.; Gac, N.; Martelli, M. OpenCL FPGA Optimization guided by memory accesses and roofline model analysis applied to tomography acceleration. In Proceedings of the 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), 2021, pp. 109–114. https://doi.org/10.1109/FPL53798.2021.00026.
30. Nguyen, T.; Williams, S.; Siracusa, M.; MacLean, C.; Doerfler, D.; Wright, N.J. The Performance and Energy Efficiency Potential of FPGAs in Scientific Computing. In Proceedings of the 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), 2020, pp. 8–19. https://doi.org/10.1109/PMBS51919.2020.00007.
31. Siracusa, M.; Delsozzo, E.; Rabozzi, M.; Di Tucci, L.; Williams, S.; Sciuto, D.; Santambrogio, M.D. A Comprehensive Methodology to Optimize FPGA Designs via the Roofline Model. *IEEE Transactions on Computers* **2021**, pp. 1–1. https://doi.org/10.1109/TC.2021.3111761.
32. Anzt, H.; Tomov, S.; Dongarra, J. Implementing a Sparse Matrix Vector Product for the SELL-C/SELL-C-o formats on NVIDIA GPUs. Technical Report UT-EECS-14-727, 2014.
33. Benner, P.; Ezzatti, P.; Quintana-Ortí, E.S.; Remón, A.; Silva, J.P. Tuning the Blocksize for Dense Linear Algebra Factorization Routines with the Roofline Model. In Proceedings of the Algorithms and Architectures for Parallel Processing - ICA3PP 2016 Collocated Workshops: SCDT, TAPEMS, BigTrust, UCER, DLMCS, Granada, Spain, December 14-16, 2016, Proceedings. Springer, 2016, Vol. 10049, *Lecture Notes in Computer Science*, pp. 18–29. https://doi.org/10.1007/978-3-319-49956-7_2.

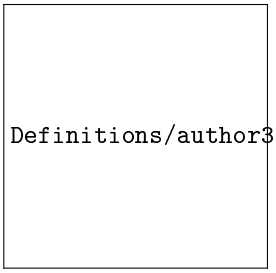
Short Biography of Authors



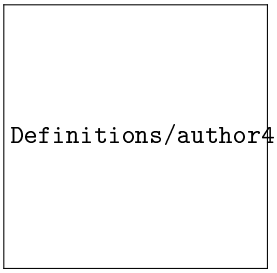
Federico Favaro Biography of first author



Ernesto Dufrechou Biography of second author



Juan P Oliver Biography of third author



Pablo Ezzatti Biography of fourth author