# An Efficient Hardware Accelerator to Handle Compressed Filters and Avoid Useless Operations in CNNs

Adrián Alcolea, Javier Olivito, Javier Resano
{alcolea, jolivito, jresano}@unizar.es

Grupo de Investigación en Arquitectura de Computadores (gaZ) Universidad Zaragoza
European Network on High Performance and Embedded Architecture and Compilation
Instituto Universitario de Investigación en Ingeniería de Aragón Universidad Zaragoza

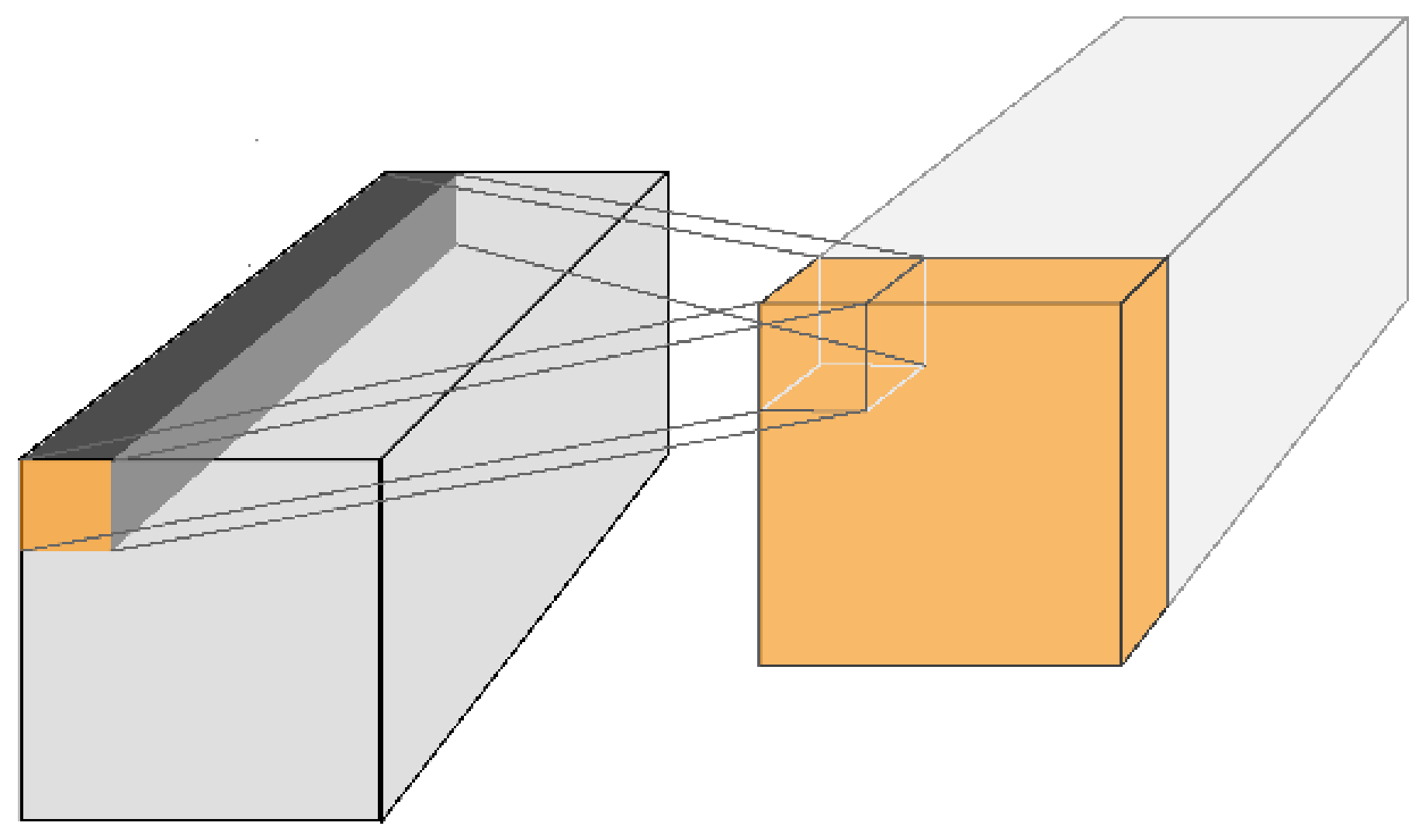
## Motivation and objectives

A significant percentage of the operations carried out in Convolutional Neural Networks (CNNs) contains a zero in at least one of their operands due to the sparsity generated by the activation functions and the pruning techniques.

The use of compressed filter matrices reduces the data that must be read from the off-chip memory, and maximizes the data that can be stored on-chip, which is very important since the access to external memories is the main energy consumption factor, and a performance bottleneck.

***Our goal is to manage compression and avoid useless operations to increase CNNs performance and energy efficiency***

## 1 How does convolution work?



Convolution is the main operation in a CNN. It consists of many multiplications between a large data matrix, the activation, and several weight matrices, the filters.

Both, filter and activation, are sparse matrices. That means they contain a lot of zeros. We will take advantage of these in two ways:

- Managing compressed filters → **Our compression scheme (2)**
- Avoiding operations with zero → **Our architecture (3, 4, 5)**

## 2 Our compression scheme

We propose a compression scheme that includes a bit for each filter value pointing out whether it is zero or not. We achieve a better compression ratio for most filters than the most common schemes.

If we take a 5 x 4 matrix with 60% sparsity and an 8-bit data size. Compression schemes with a list for the number of zeros need 2 x 8 x 8 = 128 bits, while our scheme needs 5 x 4 + 8 x 8 = 84 bits.

**Uncompressed matrix**

Sparse bidimensional matrix represented by values.

| | | | |
|---|---|---|---|
| 1 | 0 | 11 | 0 |
| 3 | 5 | 2 | 0 |
| 0 | 0 | 15 | 0 |
| 0 | 0 | 0 | 4 |
| 0 | 0 | 0 | 8 |

[5 * 4 * 8 = 160 bits]

**Variation of the CSR format**

One list stores the values

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 11 | 3 | 5 | 2 | 15 | 4 | 8 |

Another list stores the number of zeros, as an index.
The actual position is calculated from this list.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 3 | 4 | 3 |

[2 * 8 * 8 = 128 bits]

**Our matrix-index format**

One list stores the values

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 11 | 3 | 5 | 2 | 15 | 4 | 8 |

A matrix stores the position

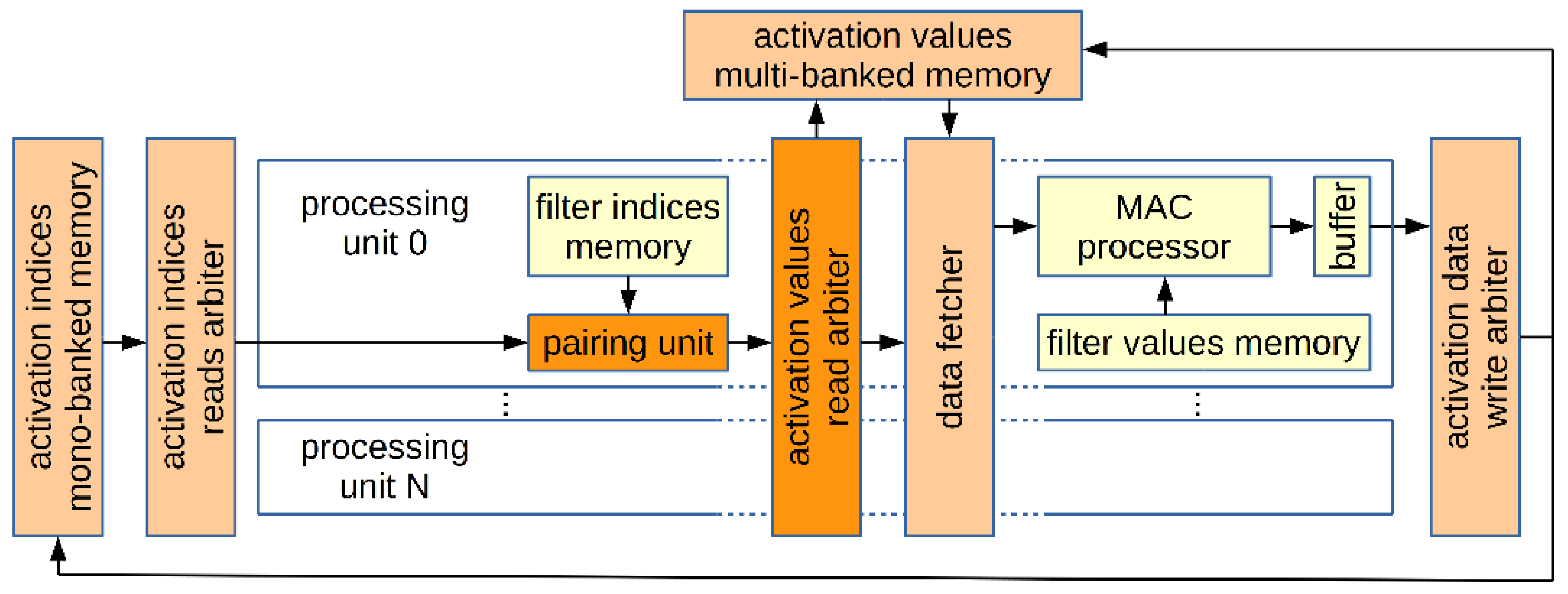
| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 |

[5 * 4 + 8 * 8 = 84 bits]

## 3 General architecture

In the proposed architecture, N convolutions are processed in parallel in N processing units.

Each of them targets a different filter and stores the compressed filter information locally, whereas the activation memories are shared.

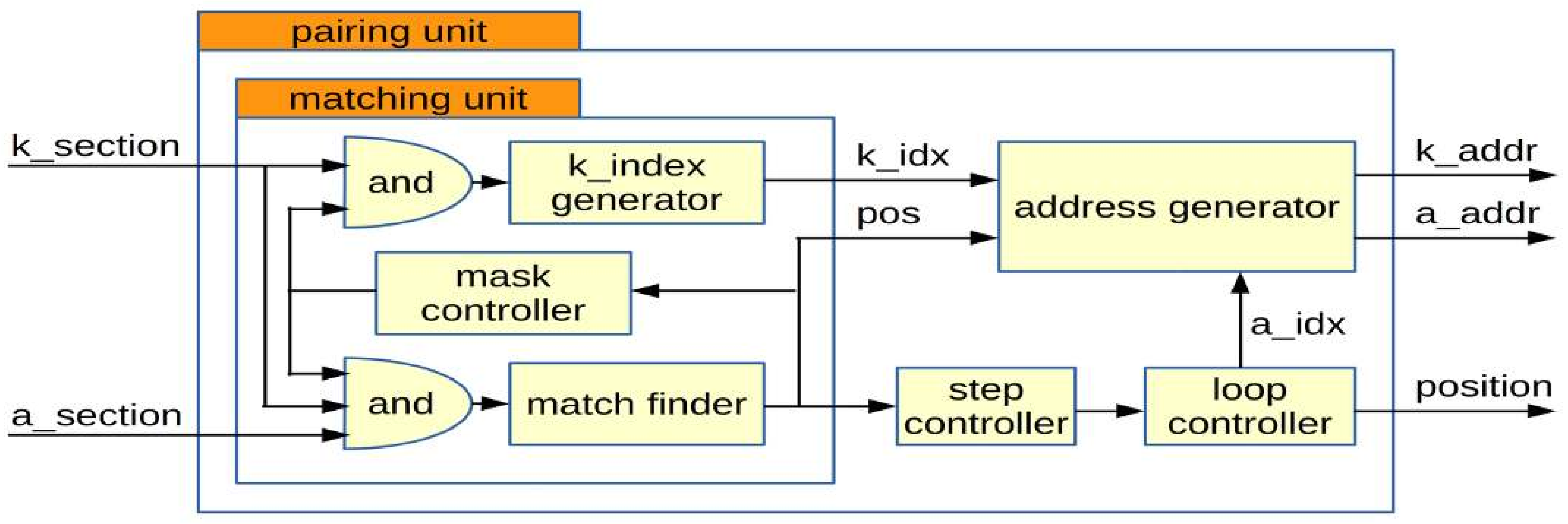

activation values multi-banked memory
activation indices mono-banked memory
activation indices reads arbiter
processing unit 0
filter indices memory
pairing unit
activation values read arbiter
data fetcher
MAC processor
buffer
filter values memory
activation data write arbiter
processing unit N

## 4 Pairing unit

This module processes the matrices of indices of the activation map and the filters, identifying which computations must be carried out (those that do not have a zero in their operands).

Its main function is to take advantage of the indices structures to efficiently find non-zero pairs.

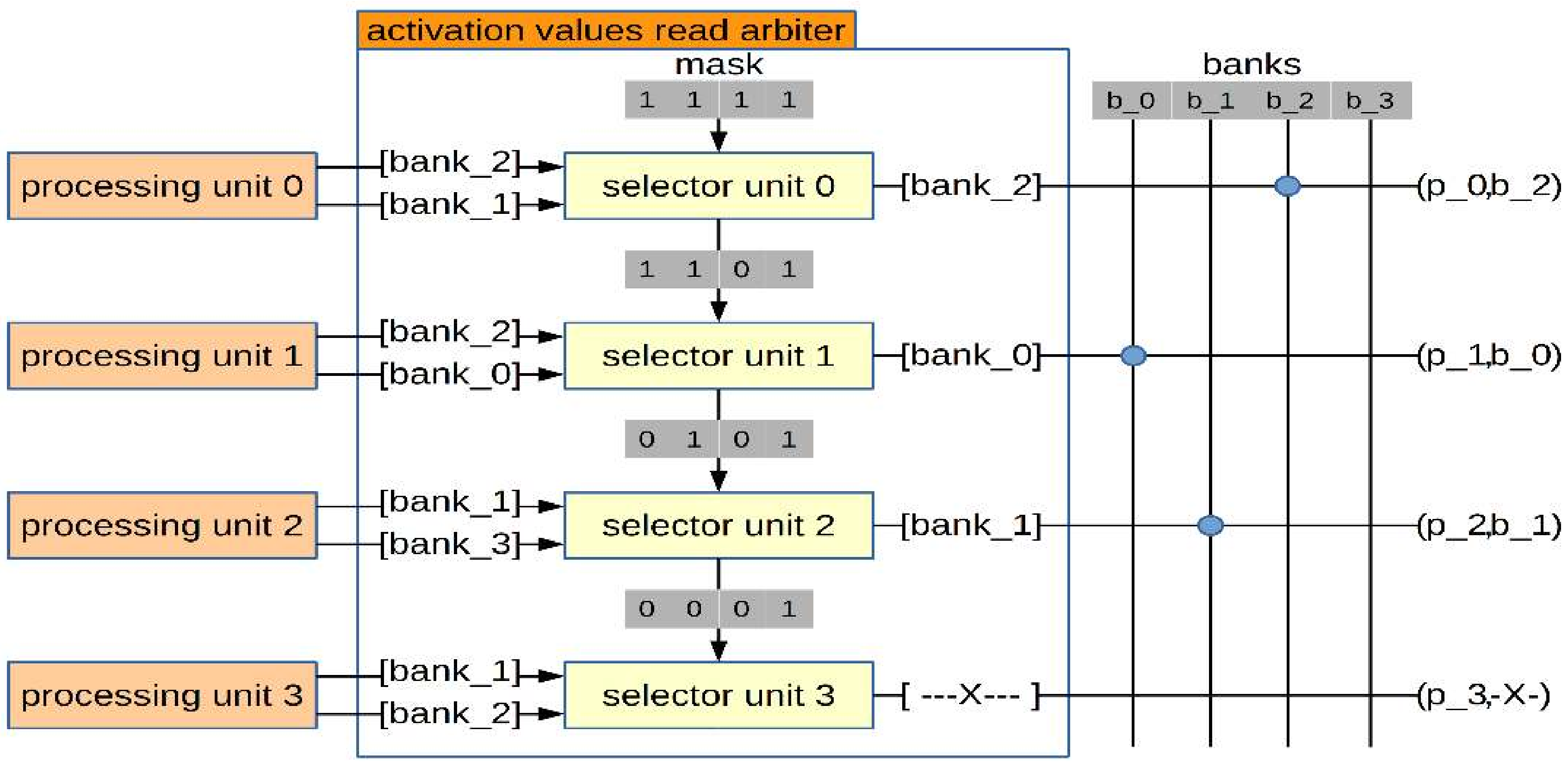
Then it uses the filter values count and the convolution loop indices to generate the actual memory addresses of the values that conform the current pair.



## 5 Activation values read arbiter

Activation values are stored in a shared memory, so access conflicts between multiple processing units may arise.

One approach to maintain the requested bandwidth consist in duplicating the number of pairs requested per processing unit, so the arbiter can take some decisions on the fetching order.



## 6 Contributions

- Our compression scheme can be used to efficiently pair the non-zero data.
- It also achieves better compression rate than state of the art.
- The proposed hardware pipeline handles compressed filters and discard all the operations where at least one operand is zero.
- We present a real implementation on an FPGA. It allows an accurate evaluation of the performance and energy efficiency of our proposal.

This work was supported in part by grants TIN2016-76635-C2-1-R (AEI/ERDF, EU) and gaZ: T58_17R research group (Aragón Gov. and European ESF).