Impact of the Array Shape and Memory Bandwidth on the Execution Time of CNN Systolic Arrays

Eduardo Yago, Pau Castelló, Salvador Petit, María E. Gómez, and Julio Sahuquillo
Department of Computer Engineering
Universitat Politècnica de València
València, Spain
eyagov@gmail.com, paucf97@gmail.com, {spetit, megomez, jsahuqui}@disca.upv.es

***Abstract*—The use of Convolutional Neural Networks (CNN) has experienced a huge rise over the last recent years and its popularity has increased exponentially, mainly due to its application both for image recognition and certain applications related to artificial intelligence. The new applications of CNN request computing demands that are difficult to address by conventional processors.**

As a consequence, accelerators –both prototypes and commercial products– focusing on CNN computation have been proposed. Among these accelerators, those based on systolic arrays have acquired a special relevance; some examples are the Google's TPU and Eyeriss.

Current research has focused on regular squared systolic arrays and most existing work assumes that there is enough memory bandwidth to feed the systolic array with input data. In this paper we explore the design of non-squared systolic arrays and address the impact of the memory bandwidth from a performance perspective.

This work makes two main contributions. First, we found that some workloads with non-squared arrays achieve similar performance to systolic arrays twice as large, which can translate in area and/or energy benefits.

Second, we present a performance comparison varying the main memory bandwidth for current DRAM devices. The analysis reveals that main memory bandwidth has a great impact on performance and that the decision of which technology use is key for the system performance. For the 64x64 array size it is necessary to use HBM2 memory to avoid the slowdown that would introduce cheaper technologies (e.g. DDR5 and DDR4).

***Keywords*-convolutional neural networks; systolic arrays; architectural parameters; performance; main memory technology;**

I. Introduction and Motivation

In the last few years, there has been an increasing interest in Convolutional Neural Networks (CNNs) by both academia and industry in the context of many AI applications. For example, Google has managed to defeat its own reCAPTCHA [1] by using CNNs. Street View [2] is a CNN that achieves 96/100 accuracy in recognizing street numbers in the images taken by their cars. CNNs are also used to improve Google's voice recognition [3], to automate data centers [4] or to save electricity in Google data centers [5].

These advances have been possible thanks to the use of computing devices able to achieve computing and energy constraints for these CNN-based applications. In this context, accelerators based on systolic arrays have been adopted as one of the most successful solutions to optimize CNN calculations, due to the natural mapping of the local operand movement which provides high compute density and low energy consumption.

To design high-performance and efficient systolic arrays for CNN-based applications, it is very important to study the characteristics and requirements of these applications. Nowadays, there is a large spectrum of CNN applications with different numbers and types of neural network layers and with a wide variability regarding the composition of each layer. By analyzing the workloads, the designer keeps in mind the most usual cases, achieving the best performance for the existing technology limitations. Depending on the objectives, the target can be to reach the maximum possible efficiency, the best performance or a tradeoff between both.

In order to obtain the most suitable systolic array design for the wide range of CNN applications, studies on the impact of the various design parameters of the array architecture (e.g array height, array width, on-chip SRAM sizes and dataflows) must be carried out to obtain the constraints of performance and energy efficiency of the applications. For this purpose, several tools have appeared such as Eyeriss [6], a researching prototype of an energy efficient systolic array, or SCALE-Sim [7], a simulator that allows researchers to analyze the impact on performance of some array parameters such as array geometry, on-chip SRAM structures and different dataflows to perform CNN computations in the systolic array, such as *Output Stationary*, *Input Stationary*, and *Weight Stationary*.

Most of the existing or proposed systolic array accelerators implement only one dataflow due to the additional cost of implementing various dataflows on the same hardware. In this way, research efforts are focused on optimizing the architecture for a given dataflow. In contrast, the motivation of this paper is to study the impact of key array parameters such as width and height of the array, and main memory bandwidth for different dataflows, which allows us to obtain useful conclusions on the design of systolic arrays for both industry and academia. To do that, we will use the SCALE-

Sim simulator. In this simulator, array geometry and dataflow are architectural configurable parameters.

On the other hand, we detected that SCALE-Sim does not model DRAM main memory with enough detail and assumes that it provides sufficient bandwidth for the applications' bandwidth requirements. This can lead to unrealistic results due to the large amounts of data processed by CNN applications and considering the bottleneck that the main memory is in terms of access time and bandwidth [8]. This paper also addresses this problem and analyses the constraints that main memory technology limitations impose on performance.

This work makes two main contributions:

- We demonstrate that non-squared systolic arrays can, for some applications, achieve performance similar to squared arrays with double size. This is mainly due to the non-squared geometry of the matrices that represent neural networks and input data in typical CNN applications. This implies that squared arrays are not always fully utilized, motivating the need of novel approaches to share or partition systolic array hardware among several applications at run time.
- We analyze the impact of the DRAM main memory technology on CNN applications executed on systolic arrays. We show that the impact of the technology is very high and that only the most advanced technologies, such as HBM2, can provide enough bandwidth to avoid the systolic array from stalling.

The remainder of this work is organized as follows. Section II discusses some background to help reading this paper. Section III describes the simulation framework used to develop this work. Section IV presents and analyzes the experimental results. Finally, Section V presents some concluding remarks.

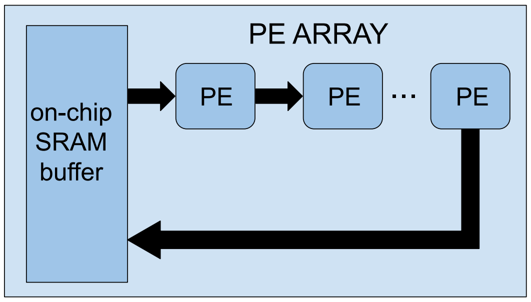
II. Background

A. Systolic Array Organization and Main System Components

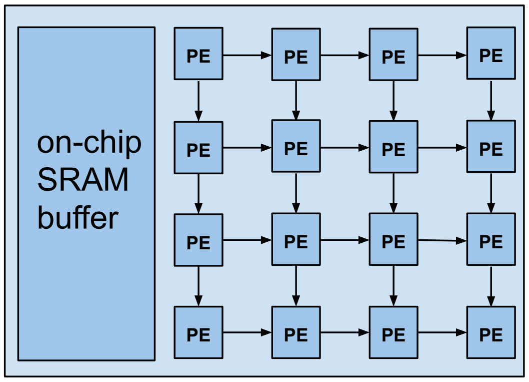
This paper focuses on the application of systolic arrays to Convolutional Neural Network (CNN) inference computation since systolic array architectures are especially suited for this type of computation [9].

In data centers, inference workloads require more system performance than that provided by CPUs. Using systolic arrays for inference increases notably the performance with respect to CPUs and GPUs as shown by the the Google TPU [10]. This is because systolic arrays provide high parallelism and allow exploiting higher data reuse than CPU and GPU devices. Below, we illustrate this parallelism and data reuse by discussing two basic systolic array organizations.

Figure 1a represents the simplest architecture of a unidimensional systolic array. The main components of this system are the on-chip SRAM buffer and the array of Processing Elements (PEs). PEs are simple compute nodes that



(a) Unidimensional array.



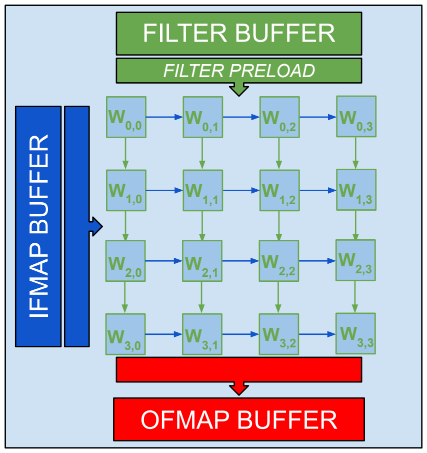
(b) Two-dimensional array.

Figure 1: Examples of systolic array organizations.

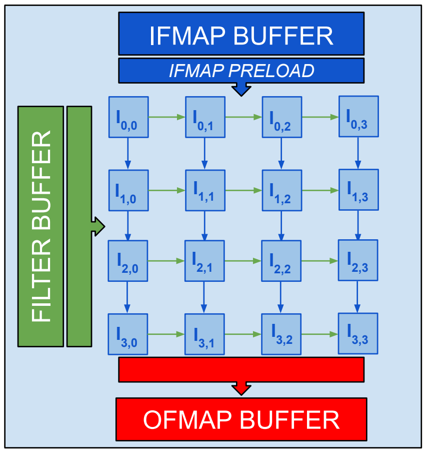
perform add and multiply operations. In the unidimensional array, they are interconnected using a unidirectional ring topology, which works as follows. The most-left PE gets its input data from the on-chip SRAM buffer, and delivers the results of its computation to the next PE neighbour in the ring, which proceeds in the same way, and so on. The last PE writes back the overall computation result to the SRAM buffer.

An improvement of the previous architecture is organizing the array in a two-dimensional way, and interconnecting PEs with a mesh network that transfers computation results in the right and down directions. Figure 1b shows an example of this architecture. In this example, a PE operates from the inputs coming from its upper and left neighbours to perform its individual computation, whose result is delivered to its right and down neighbours. An interesting observation is that the simple and small 4×4 PE array plotted in the figure can perform 16 operations in parallel per clock cycle. Thus, a typical 64×64 array can carry out up to 2^{12} operations per cycle.

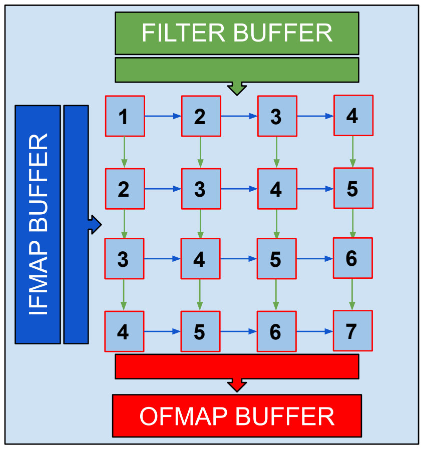
The CNN inference computation involves three matrices: i) the FILTER matrix, which contains the *knowledge* of the neural network and whose values are obtained in the CNN training process; ii) the input features matrix (IFMAP), which contains the input information for the CNN inference; and iii) the output features (OFMAP) matrix, which contains


FILTER BUFFER
FILTER PRELOAD
IFMAP BUFFER
$W_{0,0}$ $W_{0,1}$ $W_{0,2}$ $W_{0,3}$
$W_{1,0}$ $W_{1,1}$ $W_{1,2}$ $W_{1,3}$
$W_{2,0}$ $W_{2,1}$ $W_{2,2}$ $W_{2,3}$
$W_{3,0}$ $W_{3,1}$ $W_{3,2}$ $W_{3,3}$
OFMAP BUFFER

(a) Weight Stationary.


IFMAP BUFFER
IFMAP PRELOAD
FILTER BUFFER
$I_{0,0}$ $I_{0,1}$ $I_{0,2}$ $I_{0,3}$
$I_{1,0}$ $I_{1,1}$ $I_{1,2}$ $I_{1,3}$
$I_{2,0}$ $I_{2,1}$ $I_{2,2}$ $I_{2,3}$
$I_{3,0}$ $I_{3,1}$ $I_{3,2}$ $I_{3,3}$
OFMAP BUFFER

(b) Input Stationary.



(c) Output Stationary.

Figure 2: Block diagrams of the studied dataflows.

the inference results obtained by the neural network. In a systolic array architecture designed for CNN inference computing, these matrices are stored in three different SRAM buffers. Each of these buffers is named after the matrix that it stores (i.e. FILTER, IFMAP, and OFMAP buffers). The content of the input buffers (IFMAP and FILTER) is initially stored in main memory, which is also used the store the contents of the OFMAP buffer after the CNN inference computation.

To obtain the OFMAP matrix, the FILTER and IFMAP matrices are multiplied. Since these matrices are usually larger than the PE array, the computation is done distributing these matrices in smaller sub-matrices. Therefore, the overall computation is carried out in several steps, where each step works with sub-matrices to achieve the final results.

B. *Dataflows*

A dataflow defines the distribution of the matrices in sub-matrices and the mapping of input (IFMAP and FILTER) and output (OFMAP) data to each individual PE for each step. Dataflows can be classified in two main groups depending on whether they preload data in the array before starting the computation or not. The advantage of dataflows that involve preload over dataflows without preload is that, in the former, the amount of required SRAM banks is halved. Nevertheless, implementing preload increases the implementation costs [7].

In the preload process, each PE loads a component of the corresponding sub-matrix before performing its respective computations. Depending on the dataflow, the preloaded sub-matrix is obtained from the FILTER or the IFMAP matrix. In the former case, the dataflow is named Weight Stationary (WS), and if the latter case it is called Input Stationary (IS) [6].

In the WS dataflow, each component (weight) in the filter sub-matrix is mapped to a PE, as it is shown in Figure 2a. The mapping process works by transmitting the weights from the FILTER buffer at the top of the PE matrix and finalizes when all PEs have their assigned weight. Once the weights have been mapped, the input components are streamed from the IFMAP buffer. For each PE and cycle, the input components are multiplied by the preloaded weights. Partial multiplication results are stored in the PEs for further reuse [11], which includes a reduction process that is carried out by transmitting and adding the partial results along the PE columns to store the final results in the OFMAP buffer. Once all the calculations that involve the preloaded weights of a given filter sub-matrix have been completed, a subsequent filter sub-matrix is preloaded to continue performing the CNN inference computation.

The IS dataflow (Figure 2b) is similar to the WS dataflow discussed above. In this case, the IFMAP components are preloaded from the top and the components from the filter matrix are streamed from the left side.

The last dataflow studied in this work is the Output Stationary (OS), which does not perform preload (see Figure 2c). In this dataflow, each PE is devoted to compute a different component of the OFMAP sub-matrix, and it is fed with the required data to perform its corresponding computation.

Data reuse is achieved propagating input and weight data among neighbor PEs. Each cycle each PE transfers one IFMAP component to its right neighbor and one FILTER component to the neighbor below. The transferred components are required by the PEs to calculate its assigned OFMAP component, which saves memory accesses. Figure 2c shows the cycle when each PE can begin its computation starting in cycle 1. During the first cycle, only the upper right PE in the figure can compute. Then, in the second one, following the propagation procedure explained before,

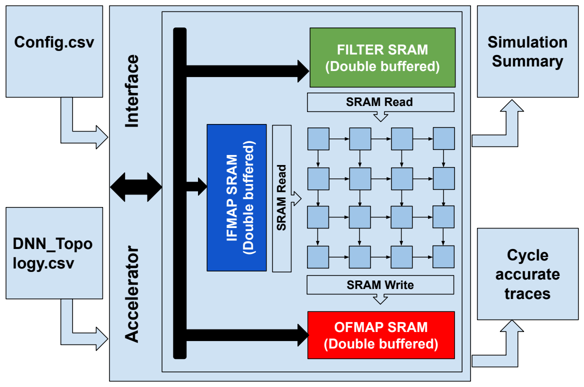


Figure 3: SCALE-Sim simulator connected to the input (config and workload csv) and output (simulation summary and cycle accurate traffic traces) files.

the two neighbors of the upper right PE can start the computation of their assigned OFMAP component, and so on. For the PE array plotted in the figure, the last PE starts its computation at cycle 7. For a generalized PE array with M and height N, the last PE starts at cycle $M + N - 1$.

III. SCALE-Sim Simulator

This section summarizes the main features of the simulator used in this paper to evaluate different systolic architectures. In particular we use SCALE-Sim (systolic CNN accelerator simulator). SCALE-Sim is a cycle-accurate simulator of CNN accelerators. The simulated accelerator is based on the systolic array architecture, similar to the one used in Google's TPU [10]. SCALE-Sim only simulates convolutional neural networks (CNN) or fully connected neural networks.

To perform a simulation, SCALE-Sim needs two input files, as depicted in Figure 3. The first file, namely `config.csv`, contains architectural parameters such as the array height, array width, the sizes of the IFMAP, FILTER and OFMAP buffers, and the selected dataflow. The second file, namely `DNN_Topology.csv`, contains the weights of the neural network and the inputs to be computed in the array. A neural network can be composed of several layers and each one requires a distinct inference computation.

As simulation results, SCALE-Sim provides, for each layer of the neural network: i) the main memory bandwidth requirement to avoid the systolic array from stalling, ii) the total execution time, and iii) the average PE array utilization. The two latter results are calculated assuming that the systolic array does not stall during the execution due to lack of enough main memory bandwidth. In addition, the simulator provides traces with detailed information about the accesses to the different memory structures (i.e. the SRAM buffers and the DRAM main memory).

Notice that the impact of the main memory implementation on the system performance is not considered since

Table I: CNN workloads used in the simulation studies.

Tag	Workload	# of layers
w1	AlphaGoZero	10
w2	DeepSpeech2	7
w3	FasteRCNN	47
w4	Neural Collaborative Filtering	11
w5	Resnet50	56
w6	Sentimental CNN	6

SCALE-Sim assumes that the main memory can provide enough bandwidth to avoid the systolic array from stalling, that is, that the systolic array can be fed with input data (e.g. weights and features) every cycle so that it can work at its maximum performance. In this work, we overcome this limitation by taking into account main memory constraints.

IV. Experimental Results

This section begins analyzing the impact of the array size on performance, that is, how the application execution time is reduced as the number of PEs involved in the computation is increased. These PEs are usually organized in a squared array; nevertheless, in this section we discuss how non-squared arrays can impact on performance. This fact could help on reducing the energy consumed by the application or its execution time.

Finally this section analyzes the performance impact of memory technology with respect to previous simulation studies (e.g. carried out with the baseline SCALE-Sim simulator) where main memory bandwidth constraints are not considered.

To carry out these studies, different workloads that cover a wide range of representative CNN applications have been considered. These workloads are listed in Table I. As it can be seen in the table, there is a large diversity of CNN applications, some of them using a deep structure due to the need to generate abstract knowledge with a huge amount of layers while in contrast others applications only use few layers. Moreover, two CNN applications with a similar number of layers can be notably different between them due to complexity differences among the layers. To illustrate the diversity of these workloads, Figure 4 shows the number of IFMAP components (left axis) and FILTER components (right axis) for each selected workload in thousands of components. As it can be seen these workloads present large differences in their composition.

A. *Impact of the Array Size*

This section analyzes the impact of the array size on performance considering regular squared array sizes across the three dataflows supported by SCALE-Sim: Output Stationary (OS), Input Stationary (IS) and Weight Stationary (WS). Six main sizes have been considered ranging from 8×8 up to 256×256 in powers of two. These experiments

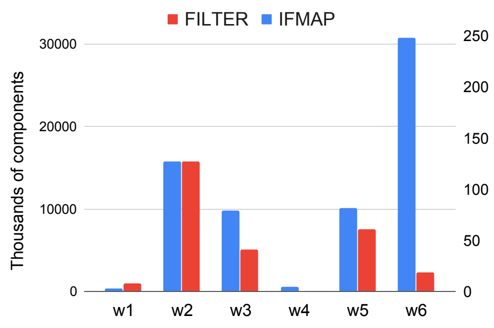


Figure 4: Component count showing the diversity of the studied workloads.

are performed varying only the array size and the dataflow while the size of the IFMAP, FILTER, and OFMAP buffers has been set to 512KB, 512KB, and 256KB, respectively.

Figure 5 plots the performance results in terms of execution time. As observed, performance significantly improves by increasing the array size from 8×8 up to 64×64 for all workloads across all the studied dataflows. However, the slope of the performance gains reduces beyond this size. Moreover, some workloads do not experience any benefit at all, such as W4 with the OS dataflow (see Figure 5a).

In summary, although intuitively, the larger the array size, the higher the achieved performance, our experimental results show that the performance gains dramatically drops in arrays with sizes beyond 64×64 and therefore using larger arrays can be inefficient since the extra non-leveraged PEs could be powered down or used for other purposes.

B. *Impact of the Array Shape*

This section analyzes the impact of the array shape on performance, that is, considering both squared and non-squared arrays across the three studied dataflows. For this purpose, the study is performed from a 32×32 array, which acts as the baseline configuration. From this array we consider two non-squared arrays obtained by doubling the width (32×64) and the height (64×32) of the baseline. Moreover, the 64×64 squared array, obtained from doubling both dimension sizes, (64×64) is also evaluated. The range between 32×32 and 64×64 has been chosen because the results of the previous section showed that beyond these array geometries increasing the array size offers diminished performance improvements.

Figure 6 shows the performance results for the four shapes considered. It can be observed that the array shape that fits better depends on each particular workload. For instance, workloads like W1, W3 and W5 with OS and WS dataflows fit better in the 32×64 configuration than in 64×32 one. Nevertheless, note these workloads improve their execution time with both the width and the height of the array, which does not justify using non-square array shapes. In contrast, we can observe that other workloads fit better to non-square shapes such as 64×32. This is the case of W2 and W6 with OS and WS dataflows. W6 improves its execution time with a 32×64 array but the improvement is very slight when compared to the improvement reached with the 64×32 configuration. That is, increasing the array width beyond a certain size stops providing significant performance improvements. Moreover, some workloads are able to obtain the same execution time with half the number of processing elements. This is the case of W2 with OS and WS dataflows, W4 with WS and IS, and finally W6 with WS for 64×32 arrays, which reach nearly the same execution time that the one obtained by the 64 × 64 arrays.

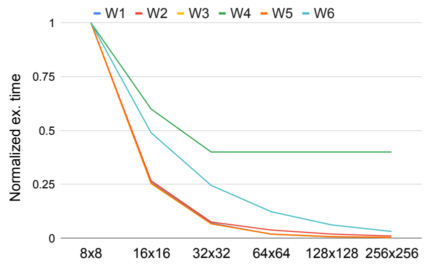
This behaviour can be explained by examining the utilization of the PEs in the array during execution. Figure 7 plots the PE utilization for W2 with the WS dataflow. As it can be seen the figure, for the 32×32 array the utilization is high (over the 75%). However, when the number of columns is doubled (32×64), the utilization is reduced by a half. If the number of rows is doubled instead (64×32), then the utilization significantly increases, which leads to the large reduction in the execution time. Finally, if both rows and columns are doubled the utilization is again reduced, which means that the application does not leverage the additional PEs. Therefore, depending on the workload, the non-utilized part of a square PE array can be switched off or assigned to other workloads to increase utilization.

Figure 6 also shows that when considering non-square shapes, the dataflow plays a relevant role on performance. First, for OS and WS a major improvement on execution time appears in the majority of the workloads when the height is increased, while for IS this improvement is due to the width increase. This is because the number of IFMAP components is always much larger than the number of FILTER components (see Figure 4) and since in IS the preloaded data are IFMAP components (see Figure 2b), this dataflow gets higher improvements by adding more columns.

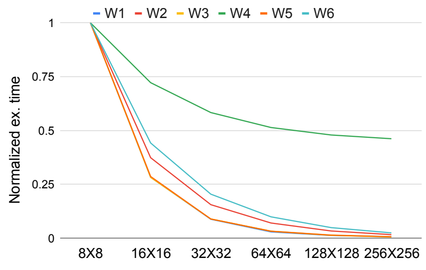
In summary, the analysis of the results shows that to design more efficient accelerators it is important to partition the array considering the utilization performed by each application. This utilization depends on the dataflow and the different sizes and shapes of the input matrices.

C. *Analysis of Memory Constraints on Performance*

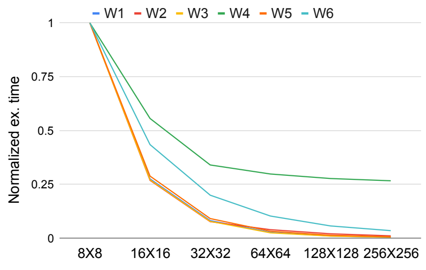
As mentioned above, SCALE-Sim does not model the main memory bandwidth technology and its constraints. Therefore, the obtained results can be too optimistic. This section estimates the execution time slowdown due to stall cycles that different main memory technologies would introduce due to bandwidth limitations. As mentioned above, the simulator provides as one of its results the main memory bandwidth requirements of each application. From this information, we can estimate the slowdown with respect to the SCALE-Sim ideal memory. The estimations are done assuming the worst possible case, where the three buffers



(a) Output Stationary.

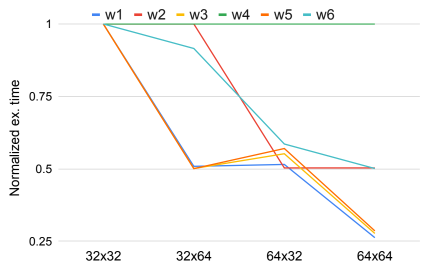


(b) Input Stationary.

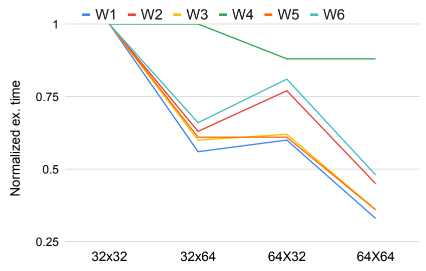


(c) Weight Stationary.

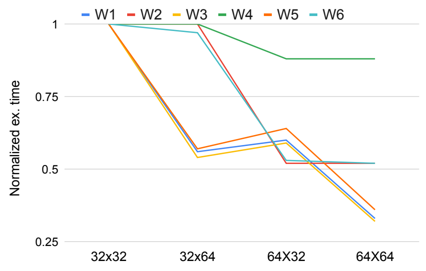
Figure 5: Normalized execution time for regular squared arrays varying the array size over a 8x8 arrays.



(a) Output Stationary.



(b) Input Stationary.



(c) Weight Stationary.

Figure 6: Normalized execution time comparison between squared arrays and non-squared arrays over 32x32 arrays.

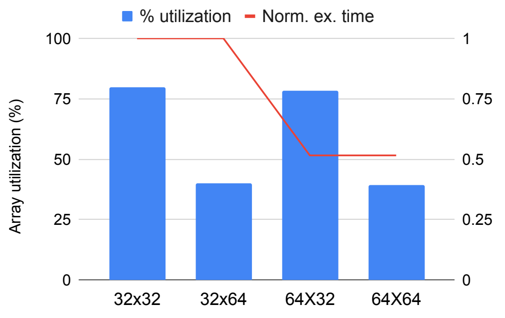


Figure 7: PE array utilization (in percentage) in W2 with WS dataflow.

Table II: Main memory technologies studied.

Technology	BW in GB/s
HBM2 [12]	256
DDR5-6400 [13]	51.2
DDR4-3200 [14]	25.6
DDR4-2666 [14]	21.3

(IFMAP, FILTER, and OFMAP) are accessed at the same time, which requires the maximum possible bandwidth.

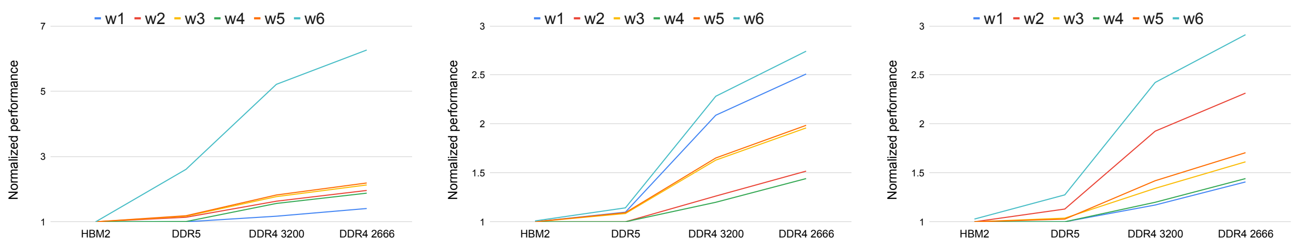
To perform these experiments, we select the main memory technologies shown in Table II, which also presents the maximum bandwidth that each technology is able to provide.

Figure 8 shows the performance slowdown normalized to the ideal configuration, that is, the one modeled in the simulator. Using HBM2 technology for implementing the main memory is able to reach the ideal execution time for almost all workloads in all dataflows except in few of them. In W6 with the IS dataflow, the execution time is increased by 1% and in W6 with the WS dataflow it is increased by 3%. In other words, for nearly all the simulations performed using HBM2 all or almost all stall cycles are avoided.

If DDR5 technology is used, the performance is impacted in all the dataflows but specially for some workloads and some dataflows. As it can be seen in Figure 8a, the execution time is increased around 10-15% for most of the workloads except for W6 where the execution time is increased by 280%. For the IS dataflow, as it can be seen in Figure 8b, most of the workloads increase their execution time around 15%, except W2 and W4, which keep the same execution time compared to the one obtained with HBM2. Finally, for the WS dataflow, as it can be seen in Figure 8c, the execution time is increased between 10-30% in the different workloads.

Figure 8 also shows that execution time slowdown obtained when using DDR4 technology. As can be seen for some workloads like W2 and W6, the degradation with this technology can reach up to 200%. For the remaining workloads the execution time increases around 30%.

In short, the memory bandwidth highly impacts on the performance of the systolic array. Depending on the final purpose of the systolic array there are different design options, one targeting implementation costs will choose DRR4 memories at a higher execution time. If the target is



(a) Output Stationary. (b) Input Stationary. (c) Weight Stationary.

Figure 8: Performance slowdown over the ideal execution time (no memory stall cycles), using different main memory technologies in a 64x64 array.

execution time no matter the implementation costs, HBM2 memories should be selected. An intermediate point may be the most interesting option, such as implementing DDR5 memories for a good tradeoff between cost and performance.

V. Conclusions

Current research on systolic array has focused on regular squared systolic arrays and assumes that there is enough memory bandwidth to feed the array with input data. In this regard, this paper makes two main contributions.

First, we found that some workloads with non-squared systolic arrays achieve similar performance to squared arrays with double size, which could translate in area and/or energy or performance benefits. In particular, depending on the workload and the implemented dataflow, benefits are obtained from increasing the number of rows or columns. As shown in this paper, this is related to the PE utilization performed by the workloads and the dataflows, which, in turn, depends on the geometries and sizes of the matrices that represent neural networks and input data in typical CNN applications. This conclusion opens a new research space. For instance, assuming squared arrays, software schedulers can be devised to share the array among different applications in order to improve the array utilization.

Second, we analyze how the memory bandwidth available in the systolic array can impact on performance. The analysis reveals that the main memory bandwidth has a great impact on performance. Thus, the main memory technology drives a tradeoff between performance and implementation costs. For instance, the performance slowdown range from 20% up to 600% with DDR4, and from 0% up to 180% when using DDR5 depending on the analyzed workload and dataflow. In contrast, the HBM2 memory technology is able to avoid these slowdowns, although at much higher implementation costs.

Acknowledgment

This work has been supported by Ministerio de Ciencia, Innovación y Universidades and the European ERDF under Grant RTI2018-098156-B-C51.

References

[1] C. S. Barr, *Researchers defeat Google's reCAPTCHA system with a 70% success rate*, 2016 (accessed October 24, 2019). [Online]. Available: https://www.slashgear.com/researchers-defeat-googles-recaptcha-system-with-a-70-success-rate-07435300/

[2] I. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[3] G. A. Blog, *The neural networks behind Google Voice transcription*, 2015 (accessed October 24, 2019). [Online]. Available: https://ai.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html

[4] S. Salman, C. Streiffer, H. Chen, T. Benson, and A. Kadav, "Deepconf: Automating data center network topologies management with machine learning," in *Proceedings of the 2018 Workshop on Network Meets AI & ML (NetAI)*, 2018.

[5] E. Richard and G. Jim, *DeepMind AI Reduces Google Data Centre Cooling Bill by 40%*, 2016 (accessed October 24, 2019). [Online]. Available: https://deepmind.com/blog/article/deepmind-ai-reduces-google-data-centre-cooling-bill-40

[6] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, pp. 127–138, 2016.

[7] A. Samajdar, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic CNN accelerator," *Computer Resesarch Repository (CoRR)*, vol. abs/1811.02883, 2018.

[8] K. K. Chang, "Understanding and improving the latency of dram-based memory systems," *Computer Resesarch Repository (CoRR)*, vol. abs/1712.08304, 2017.

[9] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[10] N. P. Jouppi and *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[11] ankur6ue, *Understanding Matrix Multiplication on a Weight-Stationary Systolic Architecture*, 2018 (accessed October 24, 2019). [Online]. Available: http://www.telesens.co/2018/07/30/systolic-architectures/

[12] G. R. Voskuilen, A. Gimenez, I. Peng, S. Moore, and M. Gokhale, "Milestone m1 report: Hbm2/3 evaluation on many-core cpu wbs 2.4, milestone ecp-mt-1000," *Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Lawrence*, 2018.

[13] A. Shilov, *JEDEC: DDR5 to Double Bandwidth Over DDR4, NVDIMM-P Specification Due Next Year*, 2017 (accessed October 24, 2019). [Online]. Available: https://www.anandtech.com/show/11238/ddr5-to-double-bandwidth-over-ddr4-specification-due-next-year

[14] Micron, *DDR4 SDRAM UDIMM*, 2016 (accessed October 24, 2019). [Online]. Available: https://www.mouser.com/datasheet/2/671/atf4c512x64az-1284569.pdf