Comprehensive Evaluation of OpenCL-Based CNN Implementations for FPGAs

Ricardo Tapiador-Morales[1], Antonio Rios-Navarro[1], Alejandro Linares-Barranco[1], Minkyu Kim[2], Deepak Kadetotad[2], and Jae-sun Seo[2]

[1] Robotic and Technology of Computers Lab, University of Seville, Seville, Spain
alinares@atc.us.es

[2] School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ, USA
jaesun.seo@asu.edu

Abstract. Deep learning has significantly advanced the state of the art in artificial intelligence, gaining wide popularity from both industry and academia. Special interest is around Convolutional Neural Networks (CNN), which take inspiration from the hierarchical structure of the visual cortex, to form deep layers of convolutional operations, along with fully connected classifiers. Hardware implementations of these deep CNN architectures are challenged with memory bottlenecks that require many convolution and fully-connected layers demanding large amount of communication for parallel computation. Multi-core CPU based solutions have demonstrated their inadequacy for this problem due to the memory wall and low parallelism. Many-core GPU architectures show superior performance but they consume high power and also have memory constraints due to inconsistencies between cache and main memory. OpenCL is commonly used to describe these architectures for their execution on GPGPUs or FPGAs. FPGA design solutions are also actively being explored, which allow implementing the memory hierarchy using embedded parallel BlockRAMs. This boosts the parallel use of shared memory elements between multiple processing units, avoiding data replicability and inconsistencies. This makes FPGAs potentially powerful solutions for real-time classification of CNNs. In this paper both Altera and Xilinx adopted OpenCL co-design frameworks for pseudo-automatic development solutions are evaluated. A comprehensive evaluation and comparison for a 5-layer deep CNN is presented. Hardware resources, temporal performance and the OpenCL architecture for CNNs are discussed. Xilinx demonstrates faster synthesis, better FPGA resource utilization and more compact boards. Altera provides multi-platforms tools, mature design community and better execution times.

Keywords: Deep learning · Convolutional Neural Network · Hardware acceleration · OpenCL · FPGA · Caffe · Xilinx · Altera

1 Introduction

In recent years, throughout a series of breakthrough algorithms [1–5], convolutional neural networks significantly improved the state-of-the-art in large-scale image recognition tasks. Driven by such success, CNNs have become widespread across a broad range of applications including vision, object detection, speech recognition, autonomous driving, image captioning, etc.

Typically CNNs consists of a large number of deep layers, and could involve hundreds of millions of parameters (i.e. convolution kernels, bias). Using high-end GPGPUs (General Purpose Graphic Processing Units), the networks are trained iteratively using back-propagation algorithm for days or weeks, and then the networks with trained weights can be deployed onto hardware for classification tasks. There has been a number of prior works [6–12] that built hardware on different platforms for efficient CNN implementation (as accelerators or complete architecture on hardware), such as FPGA [6–9] and ASIC (application-specific integrated circuits) [10–12]. ASIC or custom chip designs show better energy-efficiency, but may not flexibly map various CNN algorithms easily with the rigid circuits. On the other hand, FPGA platforms are much more flexible and could easily map any given CNN algorithm with hardware optimizations. For FPGAs, the designers could perform manual RTL designs [7], but using high-level synthesis tools could prove effective [8,9] in terms of design time and wide design space exploration. The authors in [8] employed HLS tools in Xilinx framework to optimize CNN implementation, while the authors in [9] explored Open Computing Language (OpenCL) based implementation in Altera framework for throughput optimization of CNNs.

Since the high-level synthesis tools are developed differently within different frameworks of Xilinx and Altera, it is difficult to determine which option or FPGA chip would be the best candidate for certain objectives (area, speed, etc.) from the designers point of view. In this paper, we provide a comprehensive evaluation and comparison of the same CNN using both Xilinx and Alteras OpenCL-based high-level synthesis tool flows.

The remainder of the paper is organized as follows. In Sect. 2, the OpenCL programming and models are described. In Sect. 3, Alteras OpenCL design flow and hardware system is discussed, while Xilinxs SDAaccel design flow and hardware platform is presented in Sect. 4. LeNet-5 ConvNet [13] for MNIST database digits classification scenario is presented in Sect. 5. In Sect. 6, the hardware results and implementation are compared between the two designs from different vendors in a comprehensive manner. The paper is concluded in Sect. 7.

2 OpenCL for FPGA

Parallel computing has considerably improved in the last years thanks to the technology scaling favors. In the past decade, parallelism improvement started to be oriented to multi-core architectures for general purpose computers, or many-core to more specific solutions, as GPGPUs. For example, the Tesla K80 accelerator has 4,992 cores with a dual-GPU design that allows up to 2.9 double precision TFLOPS or 8.73 single-precision TFLOPS [14]. Special interest has existed for the FPGAs and SoC framework that includes programmable logic cells oriented to embedded co-design solutions.

To implement a given CNN model onto FPGA hardware, we start from the publicly available codes in the Caffe framework [15]. The input image for the CNN model is converted to a text file from Python interface in Caffe and the text file is read from OpenCL host code. Using the Python interface in Caffe, both the input data and weights are extracted and fed to the OpenCL host code, on a batch of input images. The hardware implementation computes till the last inner product layer output and compares it to the expected output from Caffe, to ensure correct functionality. Typically, the CNN models in Caffe are realized using double-precision values for the nodes and the weights. Considering efficient hardware implementation, we first find out how much precision reduction could be achieved while having minimal degradation in the final classification accuracy, and this reduced precision will be used when the OpenCL codes are written.

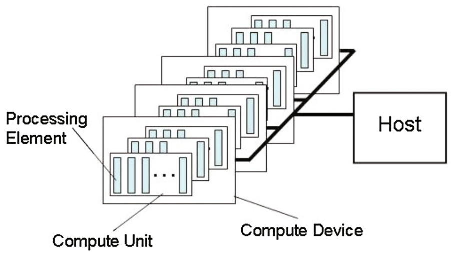


Fig. 1. OpenCL platform model.

The OpenCL *platform model* [16] consists of a host computer connected to several devices. Each OpenCL device is divided into compute units (CUs). Each CU is divided into processing elements (PE), where computations occur (Fig. 1). The OpenCL application is implemented as both host code and device kernel code. Each part will run in their specific hardware. The host code submits each kernel code as commands from the host to PEs through the memory hierarchy.

Regarding to the *execution model*, OpenCL has two units of execution: kernels that execute in one or more platforms, and a host program that executes on a host computer. Kernels execute the computation through work-items (with an associated ID), which are executed in groups (work-groups). The context of what the kernel executes is defined by the host. The host program uses the OpenCL API to create and manage the context. This API has a set of functions that enable the host interaction with devices through a command-queue. There are three main commands: kernel (to order the kernel execution), memory (data transfer between host and devices) and synchronization (synchronize points for order definition between commands). When a kernel-enqueue command submits a kernel for execution, an index space is defined. This index space is called NDRange in OpenCL, which corresponds to an N-dimensional index space. N is 1, 2 or 3. The NDRange is decomposed into work-groups forming blocks. It is defined by three integer arrays: global size (the extent of the index space in each dimension), an offset index (initial value of indices in each dimension), and the local-size (size of the work-group in each dimension).

The *memory model* consists of four regions in OpenCL for work-item execution. *Global memory* is where all work-items of all work-groups have to read and write data. These accesses must be cached. *Constant memory* is a region that remains without changes during the kernel execution. The host initializes this memory. *Local memory* is the one used by work-groups locally. It is shared by all work-items. It can be mapped into regions of the global memory. And *private memory* is a memory region that is only visible for a work-item, such that any other work-item cannot access this memory of a particular work-item. Data flow between memory regions is controlled through commands that the host enqueues. The memory consistency is guaranteed in a work-item and between a work-group and all its work-items, but there is no guarantee of memory consistency between different work-groups executing a kernel.

In OpenCL, there are mainly two *ways to parallelize a kernel*: (1) using multiple compute-units (CUs) in parallel (see Fig. 2, left), and (2) vectorising data processing through SIMD kernels with a unique CU (see Fig. 2, right). When multiple CUs are used in parallel, a kernel is replicated and the replicated kernels work simultaneously, increasing throughput and, therefore, consuming global memory bandwidth and hardware resources. On the other hand, SIMD vectorization increases throughput by vectoring kernels, which allows processing multiple work items in a single instruction (SIMD). This alternative is more efficient than using several CUs because it only duplicates the data paths. In this paper, SIMD experiments are presented because the use of replicated CUs generates global memory bottlenecks due to many parallel accesses during the execution.

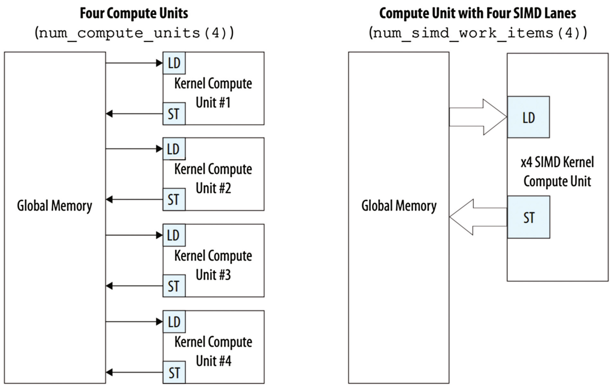


Fig. 2. Multiple parallel CU (left) versus single CU with SIMD (right).

Finally, there are two supported *programming models*: data parallel and task parallel. The data parallel model defines a computation as a sequence of instructions applied to multiple elements of a memory. On the other hand, the task parallel model requires the kernel to be executed in a single work-item of a work-group. In this case, several kernels can be executed in parallel. Synchronization is possible between work-items of a work-group or through two types of enqueued commands: barrier (it ensures all previous commands have been executed) or wait-on-an-event (the command to be executed waits for a particular event in memory before executing itself).

3 Altera OpenCL

Altera OpenCL (AOCL) is a framework for developing host applications that send kernels to be executed in parallel in FPGA resources. Work-groups, their work-items and memory models are implemented automatically in FPGA resources from an OpenCL description of the kernels and a C++ host application calling different functions from API libraries, such as: set buffers, call kernels, synchronize through events and read results. AOCL allows users to abstract the traditional hardware FPGA development flow and instead work with a much faster and higher-level software development flow. Using this design flow, it is possible to emulate OpenCL code in a FPGA, generating synthesis report files as timing or resources summaries.

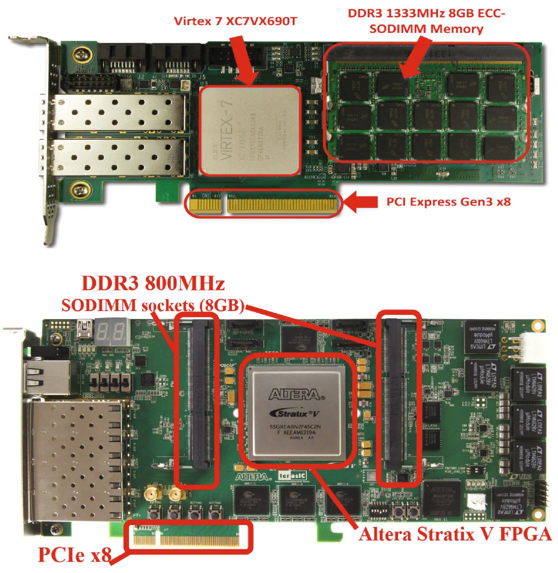

Virtex 7 XC7VX690T
DDR3 1333MHz 8GB ECC-SODIMM Memory
PCI Express Gen3 x8
DDR3 800MHz SODIMM sockets (8GB)
Altera Stratix V FPGA
PCIe x8

Fig. 3. Bottom Terasic DE5-Net Altera platform. Top Alpha-Data ADM-PCIE-7V3 Xilinx platform.

The design flow consists of two parts: host software application and kernel accelerator hardware on FPGA. The concept is that host sends data to the kernels, where complex calculations are accelerated. The design flow for an Altera board using OpenCL consists of several steps. The first step is to describe the functionality of the kernels using C/C++ and then to optimize each kernel applying OpenCL directives to generate a *.cl file. In addition, a host application must be written in C/C++ using the recommended environment. The OpenCL implementation of the LeNet-5 CNN for MNIST handwritten digit recognition [13] has been deployed on a Terasic DE5-Net (see Fig. 3). This board supports Altera OpenCL and its main characteristics include up to 8 GB DDR3 RAM memory running at 800 MHz, 72 Mb SRAM running at 550 MHz, PCIe-x8 and Altera Stratix V GX-5SGXEA7N-2F45C2 FPGA, which features are shown in Table 1.

Table 1. Right resources of Alpha-Data ADM-PCIE-7V3 Xilinx platform. Left resources of Terasic DE5-Net Altera platform

Resource	Amount	Resource	Amount
Logic elements (K)	622	Logic cells (K)	693
M20K (Blocks/Mbits)	2560/50	Slices (K)	108
18-bit 18-bit Multipliers	512	Distributed RAM(Kb)	10
27-bit 27-bit Multipliers	256	DSP slices	3600
		Block RAM (#/Mb)	2940/53

4 Xilinx SDAccel

The SDAccel [17] development environment is a command line based tool suite for compiling OpenCL programs into a Xilinx FPGA device. The design flow is similar to AOCL in terms of host and kernel descriptions. Directives and API must be replaced when same project is developed for both vendor environments. SDAccel runs on RedHat Linux OS in a batch mode using a command file. These commands allow to define the solution name, adding the target device and host files, creating the kernels and adding the files where they are implemented, creating the Xilinx OpenCL compute unit binary file, and building and packaging the systems. Several CUs per kernel can be implemented. Each CU can have several PEs, which emulates the SIMD architecture. One important advantage over the Altera tool is that SDAccel lets the programmer to test the application (emulation on CPU) before compiling and generating the FPGA binary file. A disadvantage is that SDAccel is less mature than AOCL.

SDAccel allows hardware emulation of the codesign program before building the system for FPGA. Hardware emulation is slower than CPU emulation since it uses a hardware simulator, but this emulation reproduces the final design on FPGA. The main advantage of using hardware emulation is to avoid the long implementation times (8 h on average for this work).

Table 2. Test results: No parallelism/Unroll/SIMD

Kernel Name	*Exec. time (ms)*	*Logic Cells / Elem. (K)*	*DSP slices*	*BRAM (Kb)*
		Xilinx Virtex 7 690T		
conv_pool1	3.63 / 1.96 / 1.96	4.9 / 6.2 / 5.1	11 / 11 / 11	180 / 216 / 216
conv2	7.62 / 4.92 / 4.92	4.8 / 4.8 / 4.9	11 / 11 / 11	108 / 144 / 144
pool2	0.03 / 0.06 / 0.06	3.0 / 4.0 / 3.0	4 / 4 / 4	72 / 144 / 144
ip1_relu	0.55 / 0.55 / 0.55	4.2 / 4.2 / 4.2	11 / 11 / 11	72 / 72 / 72
ip2	0.35 / 0.35 / 0.35	4.0 / 3.0 / 4.0	9 / 9 / 9	72 / 72 / 72
		Altera Stratix V GXA-7		
conv_pool1	1.01 / 1.01 / 0.98	145.7 / 42.3 / 73.7	8 / 31 / 57	5225 / 6205 / 11200
conv2	3.95 / 3.96 / 4.27	300.5 / 34.0 / 34.0	8 / 31 / 31	3207 / 4882 / 4900
pool2	0.08 / 0.07 / 0.13	6.9 / 6.9 / 6.8	2 / 2 / 2	279 / 273 / 279
ip1_relu	1.01 / 1.81 / 2.02	5.8 / 5.8 / 5.8	4 / 4 / 4	1471 / 1470 / 1500
ip2	0.15 / 0.14 / 0.13	5.7 / 5.7 / 5.7	4 / 4 / 4	1471 / 1470 / 1500

After emulation, the final step is to build the system in the real hardware of the target device. When the compilation/implementation is completed, a number of files have been created to run the application, such as the executable, the Xilinx OpenCL binary container (*.xclbin) and the FPGA programming file. For these experiments, the Alpha Data ADM-PCIE-7V3 [18] board (see Fig. 3) has been used under the CentOS 6.6 operating system. The main features include two 8 GB ECC-SODIMM memory up to 1333 MT/s (faster than Altera DE5 platform), one PCI Express Gen3 x8 and Xilinx Virtex 7 XC7VX690T-2FFG1157C. The features of the FPGA are listed in Table 2. Besides the DSP slices, the specification of the FPGA is similar to that of Alteras Stratix V-GXA7.

5 LeNet-5 and MNIST Scenario

LeNet-5 CNN [13] architecture (shown in Fig. 5) serves as the baseline for many recent CNN-based classification algorithms. It combines three architectural ideas to ensure a certain degree of shift, scale and distortion invariance: local receptive fields, shared weights and spatial sub-sampling. The input layer represents a size-normalized and centered image. In this case, the size corresponds to the size of MNIST database digits (28×28). The first layer (C1) is the result of a set of convolutions over the input image. Each pixel in C1 receives inputs from a set of units located in a small neighborhood of the previous layer. This represents the kernel of the convolution (5×5 in our case). These operations are able to learn and extract elementary visual features, such as edges, end-points, and corners. The combination of these features by subsequent layers are able to detect higher-order features. C1 in this example extracts 20 features from the input image. S2 performs a sub-sampling operation of local averaging, reducing the resolution of the feature maps where distinctive features are encoded. Typically, these convolution and sub-sampling layers are sequentially instantiated for feature map combinations. They are implemented in a bi-pyramid way: at each layer, the number of feature maps is increased as the spatial resolution is decreased. C3 is a convolution layer for 50 smaller feature maps and S4 is the corresponding sub-sampling layer that performs the same operation as that in S2. C3 combines all of the S2 features. The last layer of this CNN is a fully-connected classifier with 500 input neurons and 10 output neurons, which also includes a Rectification Linear Unit (ReLU).

6 Comparison Study

The implementation of this Le-Net5 using the OpenCL framework impose some restrictions. Figure 4 (bottom) shows the block diagram of the OpenCL solution. It can be seen that the host application, running on a computer, sends input images and kernel weights to the logic through PCIe interface. Data is then stored in the DDR memory in the platform, called Global Memory. This memory is continuously and iteratively accessed by the logic (FPGA) through all the

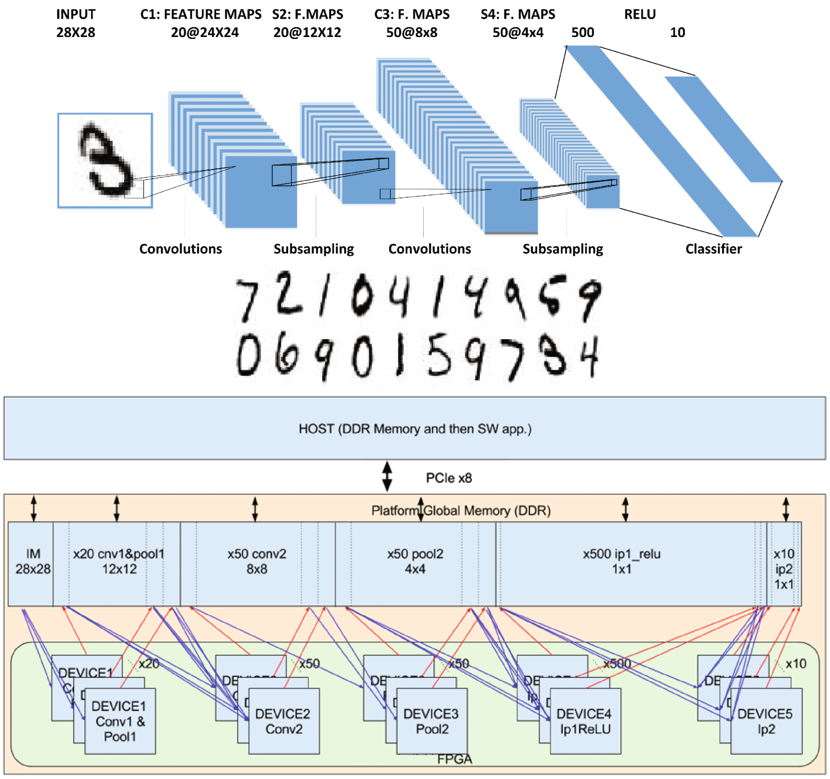


Fig. 4. LeNet-5 ConvNet architecture (top) for MNIST digit recognition (middle) and its OpenCL based hardware block diagram (bottom).

parallel devices physically implemented in hardware. The CNN is structured in 5 kernels (stages), where first kernel implements first layer convolutions and their subsampling operations (conv_pool1); second kernel performs the second layer convolutions (conv2), which is more complex since it has to take results from 20 instances of previous layer, and perform convolutions for 50 instances of this second layer. Then, the third kernel implements the second subsampling operations (pool2). The forth kernel has 500 instances for the classifier unit, whose inputs are the outputs of the previous 50 instances (ip1_relu). Each of these 500 devices send their output to a final layer with 10 instances (one per digit) to categorize the winner digit in the classification (ip2). Each of these devices read the global memory, process the corresponding operation, and then write back the results to the global memory. Consecutive kernels (stages or layers of the CNN) execute in-order, which are controlled by special events included during OpenCL compilations. This architecture needs a high bandwidth DDR memory interface to support all required parallel instances. OpenCL can implement each kernel in a replicated manner as many times in parallel as possible, or it can execute one after the other sequentially if no parallelism can be implemented. As more parallelism is employed, the global memory behavior worsens. The main difference between Altera and Xilinx platforms is the DDR3 on-board memory speed (800 MHz for Altera and 1333 MHz for Xilinx) as mentioned previously. OpenCL allows other memory implementations to avoid this shared memory bottleneck, like local pipes that connect two devices directly in the logic. Each of these pipes is implemented through small FIFOs as a point-to-point communication channel between two devices. For CNNs, these pipes do not represent a feasible solution because internal convolution layers, such as C3 in this case, have to read all the S2 outputs and combine them into each of the 50 C3 outputs. This represents 50 pipes at C3 per for each of the 20 S2 units, which is not viable, in terms of resource consumption, for the selected platforms. Therefore, we selected the global memory interface as the possible solution to work for both platforms, and we provide a comprehensive comparison. Three different tests have been developed for these platforms with the Le-Net5. The first test consists of comparing each FPGA executing each layer of the CNN without any kind of parallelism. The second test aims to do same measurements when loops are unrolled. For the last test, SIMD directives have been included to vectorise each layer. Table 3

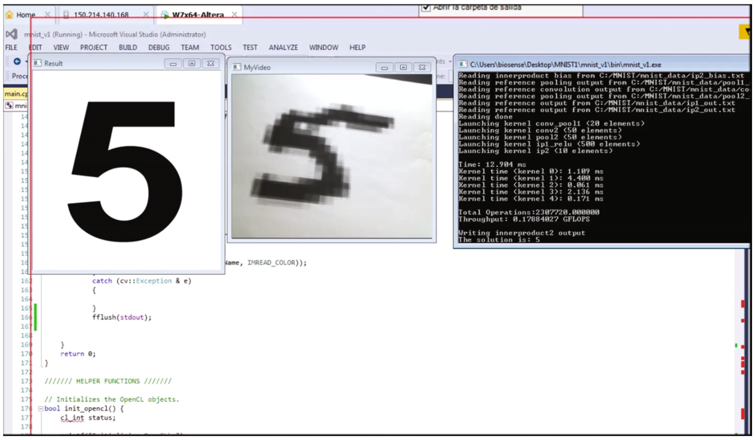
Table 3. Acceleration comparison

Kernel Name	***Xilinx vs Altera Acceleration***	***% Acceleration***
conv_pool1	3.59 / 1.94 / 2	259 / 94 / 100
conv2	1.92 / 1.24 / 1.15	92 / 24 / 15
pool2	-2.66 / 1.16 / 2.16	-166 / -16 / -116
ip1_relu	-1.83 / 3.29 / 3.67	-83 / -229 / -267
ip2	2.33 / 2.5 / 2.69	133 / 150 / 169

shows the results of these three experiments. Execution times, logic resources, the number of DSP units and needed blockRAM are shown per kernel. In general, execution time is improved upon employing more parallelism up to a limit. The limit occurs due to the bottleneck that the global memory accesses impose. As expected, the usage of logic gates and DSP units increases when parallelism is increased. Altera tools are able to extract much more parallelism than Xilinx, as it can be seen on logic elements/cells and DSP utilization. There are very small differences between unrolling and SIMD for both platforms for this experiment. Altera tool is able to extract more aggregation for SIMD than Xilinx. In fact, for Xilinx, both unrolling and SIMD have almost same results.

7 Real Time Experiment

In order to demonstrate the different DDR memory bandwidth limits of these two platforms, the same real-time experiment has been performed in both platforms. The experiment consists of connecting a webcam to the host application, which continuously reads in an image frame, normalizing it and resizing to 28×28 pixels using OpenCV libraries. The host sends kernels parameters in the beginning and then it iterates the process of acquiring an image frame, pre-processing it, sending it to the platform and checking the final classification results. The on-board DDR in the Altera platform could not support the memory bandwidth required by this demonstration and the time per frame is continuously increasing (starting at 10 ms per frame). In contrast, Xilinx platform supported this real-time experiment owing to the higher DDR bandwidth. Results show that time increases when parallelism is applied. This is due to memory bandwidth when multiple access to global memory are done. Bottlenecks slow down kernel increasing execution time. Figure 5 shows a screen-shot of the real time running


Home
150.214.140.168
W7x64-Altera
Abrir la carpeta de salida
mnist_v1 (Running) - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ANALYZE WINDOW HELP
Result
MyVideo
C:\Users\biosense\Desktop\MNIST1\mnist_v1\bin\mnist_v1.exe

```
Reading innerproduct bias from C:/MNIST/mnist_data/ip2_bias.txt
Reading reference pooling output from C:/MNIST/mnist_data/pool1_
Reading reference convolution output from C:/MNIST/mnist_data/co
Reading reference pooling output from C:/MNIST/mnist_data/pool2_
Reading reference output from C:/MNIST/mnist_data/ip1_out.txt
Reading reference output from C:/MNIST/mnist_data/ip2_out.txt
Reading done
Launching kernel conv_pool1 (20 elements)
Launching kernel conv2 (50 elements)
Launching kernel pool2 (50 elements)
Launching kernel ip1_relu (500 elements)
Launching kernel ip2 (10 elements)

Time: 12.904 ms
Kernel time (kernel 0): 1.109 ms
Kernel time (kernel 1): 4.400 ms
Kernel time (kernel 2): 0.061 ms
Kernel time (kernel 3): 2.136 ms
Kernel time (kernel 4): 0.171 ms

Total Operations:2307720.000000
Throughput: 0.17884027 GFLOPS

Writing innerproduct2 output
The solution is: 5
```

```
Name, IMREAD_COLOR));
            catch (cv::Exception & e)
            {

            }
            fflush(stdout);


    }
    return 0;
}

/////// HELPER FUNCTIONS ///////

// Initializes the OpenCL objects.
bool init_opencl() {
    cl_int status;
```

Fig. 5. LeNet-5 ConvNet architecture (top) for MNIST digit recognition (middle) and its OpenCL based hardware block diagram (bottom).

demonstration. In general, logic elements, DSP and BlockRAM have increased when parallelized directives are applied. However, the time does not get better due to bottleneck generated by DDR memory bandwidth. Table 3 represents the acceleration between vendors. Execution times for Xilinx are much better than Altera except for pool2 and ip1_relu stages.

8 Conclusions

This work presents a comparison between two OpenCL FPGA-based platforms (Altera and Xilinx) executing a convolutional neural network. Results show that the Altera platform has better execution time for each kernel than the Xilinx platform for all test scenarios. However, the Xilinx platform requires less FPGA resources than the Altera counterpart to execute the same CNN model. The real-time experiment developed for both platforms has demonstrated that the DDR memory bandwidth is crucial for the global memory communication architecture. Other memory architectures, such as pipes, were implemented internally to the FPGA without requiring any off-chip memory bandwidth, but it was insufficient for CNNs because of their point-to-point connections. A new memory model that allows having double-buffered memory spread on the FPGA blockRAM will avoid the bottlenecks identified in this work. This will allow having more CUs in parallel to further improve the performance. Therefore, beyond the differences between the platforms, this presented exploratory work for implementing full CNNs architectures on FPGA with OpenCL, shows that proposed codesing architecture lacks on memory bandwidth because of the dense connections between layers.

Acknowledgments. This work has been partially supported by Xilinx and Altera University Programs (through platform donations); by the NPP project funded by SAIT (2015–2018); and by the Spanish government grant (with support from the European Regional Development Fund) COFNET (TEC2016-77785-P).

References

1. Krizhevsky, A., et al.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (2012)
2. Simonyan, K., Zisserman, K.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)
3. Szegedy, C., et al.: Going deeper with convolutions. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
4. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning (ICML) (2015)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition (2015). arXiv:1512.03385

6. Farabet, C., et al.: Hardware accelerated convolutional neural networks for synthetic vision systems. In: IEEE International Symposium on Circuits and Systems (2010)
7. Gokhale, V., et al.: A 240 G-ops/s mobile coprocessor for deep neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops (2014)
8. Zhang, C., et al.: Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: ACM International Symposium on Field-Programmable Gate Arrays (FPGA) (2015)
9. Suda, N., Chandra, V., Dasika, G., Mohanty, A., Ma, Y., Vrudhula, S., Seo, J., Cao, Y.: Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In: ACM International Symposium on Field-Programmable Gate Arrays (FPGA) (2016)
10. Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., Temam, O.: DaDianNao: a machine-learning supercomputer. In: IEEE/ACM International Symposium on Microarchitecture (MICRO) (2014)
11. Chen, Y.-H., Krishna, T., Emer, J., Sze, V.: Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. In: IEEE International Solid-State Circuits Conference (ISSCC) (2016)
12. Sim, J., Park, J.-S., Kim, M., Bae, D., Choi, Y., Kim, L.-S.: A 1.42TOPS/W deep convolutional neural network recognition processor for intelligent IoE systems. In: IEEE International Solid-State Circuits Conference (ISSCC) (2016)
13. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
14. Tesla K80 GPU Accelerator, Board Specification BD_07317_001_V05, January 2015. http://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf
15. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. In: ACM International Conference on Multimedia (2014)
16. Howes, L., Munshi, A. (eds.) The OpenCL Specification Version: 2.0. Khronos OpenCL Working Group. https://www.khronos.org/
17. Xilinx SDAccel development environment user guide. http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html
18. Alpha Data ADM-PCIE-7V3 user manual. http://www.alpha-data.com/pdfs/adm-pcie-7v3%20user%20manual.pdf