FPGA Based Cellular Neural Network Optimization: From Design Space to System

Zhongyang Liu
Zhejiang University

Shaoheng Luo
Zhejiang University

Xiaowei Xu
University of Notre Dame

Yiyu Shi
University of Notre Dame

Cheng Zhuo*
Zhejiang University

ABSTRACT

Cellular Neural Network (CeNN) is considered as a powerful paradigm for embedded devices. Its analog and mix-signal hardware implementations are proved to be applicable to high-speed image processing, video analysis and medical signal processing with its efficiency and popularity limited by smaller implementation size and lower precision. Recently, various digital implementations of CeNNs on FPGA have attracted researchers from both academia and industry due to its high flexibility and short time-to-market. However, most existing implementations are typically bounded utilizing the advantages of FPGA platform inadequately with unnecessary design and computational redundancy that prevents speedup. To address these issues, we propose a multi-level optimization framework for energy efficient CeNN implementations on FPGAs. In particular, the optimization framework is featured with three level optimizations: system-, module-, and design-space-level, with focus on computational redundancy and attainable performance, respectively. Experimental results show that with various configurations our framework can achieve an energy efficiency improvement of 3.54× and up to 3.88× speedup compared with existing implementations.

CCS CONCEPTS

• **Hardware** → **Reconfigurable logic and FPGAs**; *Emerging architectures*;

KEYWORDS

Cellular neural network, FPGA, acceleration

ACM Reference Format:
Zhongyang Liu, Shaoheng Luo, Xiaowei Xu, Yiyu Shi, and Cheng Zhuo. 2017. FPGA Based Cellular Neural Network Optimization: From Design Space to System. In *NCS '17: Neuromorphic Computing Symposium, July 17–19, 2017, Knoxville, TN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3183584.3183619

*Corresponding Author: czhuo@zju.edu.cn

NCS '17, July 17–19, 2017, Knoxville, TN, USA

ACM ISBN 978-1-4503-6442-3/17/07…$15.00
https://doi.org/10.1145/3183584.3183619

1 INTRODUCTION

With an increasing popularity of smart sensing and growing low-power demands for IoT applications, conventional image processing designs suffer from inefficient data processing and high power consumption. Cellular Neural Network (CeNN) is considered as a viable option to improve the efficiencies in both power and performance. CeNN is inspired by the functionality of neurons through modeling the working principles human brain sensing. It has unique potentials at various image processing areas such as noise cancellation [11], edge detection [6], path planning [8] and segmentation [5]. Recently, CeNN has aroused interests from both academic and industry [1][20][12] for efficient hardware implementations.

The structure of CeNN makes it a natural fit for analog implementations. Many studies have discussed possible analog design details [7][18][13][2] to achieve high performance with fast convergence rate and the convenience of integration with image sensors. However, due to real time response constraints and complexity from various application scenarios, there are still a few key challenges that need to be well addressed in the conventional CeNN analog implementations:

- **I/O Resource Consumption:** Each input is required to correspond to a unique neuron cell, consuming too many I/O resources. For example, a recent implementation in [2] can only support 256×256 pixels, which is far below the demands from mainstream images specification, *e.g.*, 1920×1080 pixels.
- **Noise Susceptibility:** Analog circuits are prone to noise, thereby limiting the output data precision to 7 bits or below [19]. Thus, such analog implementations can hardly process regular 8-bit gray images.

In view of the aforementioned issues, digital implementations of CeNNs have become a popular option [10][12] at the cost of data approximation. However, for such digital implementations, their efficiencies are limited due to the increased number of iterations for discrete approximation. For example, in order to process an image of 1920x1080 pixels, it requires 4-8 Giga operations (for 3x3 templates with around 39 operations per pixel and 50-100 iterations), which can hardly be completed to meet the typical real-time video streaming requirement of 40ms.

To tackle the computational challenge, CeNN accelerations on digital platforms such as ASICs [10][12], GPUs [17] and

FPGAs [3][16] [14][19][20] [15] have been explored. Among them, FPGA is a popular alternative due to its high flexibility and low time-to-market. The work [3] presented a baseline design of FPGA implementation of CeNN for several common image processing applications. The study [16] took the advantage of reconfigurable computing for CeNNs. The proposed implementation supported multi-scale cellular image processing as well as several pattern recognition applications. Recently, a CeNN implementation for binary image processing was demonstrated in [15], which focused on highly efficient processing on binary operations to achieve 4.43× speed-up. Expandable and pipelined implementations were also proposed on multiple FPGAs [14] to further reduce the computational cost. Based on that, the work [19] implemented a high throughput CeNN system to achieve real-time video streams processing at a visible pixel rate of 124.4 Mpix/s.

With all these works sharing the same architecture for CeNN and various acceleration techniques, there still lacks a comprehensive study and design exploration framework for efficient CeNN implementation, which covers the following remaining issues:

- First, only a single processing element is used to compute one iteration in many existing works. In other words, the workload cannot be segmented in temporal domain, and hence do not exploit the full potential of parallelism [19].
- Second, many works are unaware of the repeated parameters in the template and incur unnecessary I/O overheads to obtain those parameters. Since I/O access is much slower than computation, such unawareness effectively leads to computational redundancy.
- The last but not the least, most design space exploration do not well study the utilization of available resources and bandwidth of FPGAs to achieve the best performance.

It is not a trivial task to resolve the aforementioned issues. Various optimization techniques are required for scalable processing elements. Thus, in this paper, in order to address those remaining concerns, we propose a multi-level optimization framework for CeNN computation for scalability and efficiency. The framework is featured with three-level optimizations:

- System level optimization(SLO): A parallel and tilling processing (PTP) is proposed to separate workloads and enable the parallelism. Moreover, data-reuse optimization is applied to eliminate unnecessary memory usage and increase the computation efficiency.
- Module level optimization(MLO): For processing elements, the requirements of multipliers are far beyond affordable resources. Thus, a parameter quantization scheme is adopted to reduce unnecessary multiplications. Moreover, a memory access technique is proposed to make full use of memory bandwidth and reduce processing latency.
- Design space level optimization(DSLO): With different available logical resources and bandwidth, the system may achieve different performance, which however is not linear with resource and bandwidth, and hence cannot be simply predicted. As a result, a roofline model based method is adopted to conduct more rigorous performance optimization for specified FPGAs.

Experimental results with different configurations on multiple FPGA devices are investigated. The results show that, compared with existing works, the proposed optimization framework can achieve an energy efficiency improvement of 3.54× and a speed up of 3.88×.

The remainder of the paper is organized as follows. Section 2 introduces background information of the paper. Section 3 resents the framework overview. Section 4 describes system level optimization for efficient implementation. The proposed module level optimization techniques are shown in Section 5. Design space level optimization is discussed in Section 6. Experiments and result discussion are presented in Section 7, followed by the concluding remarks in Section 8.

2 BACKGROUND

2.1 Cellular neural networks

Different from the prevalent CNNs superior for classification tasks, CeNN model is inspired by the functionality of visual neurons, and a mass of neuron cells are connected with neighbouring ones. Only adjacent cells can interact directly with each other, which is a significant advantage for hardware implementation resulting in much less routing complexity and area overhead. For the widely used 2D CeNN with space-invariant templates, the dynamics of each cell state with an M×N rectangular cell array [4] are as follows:

$$\dot{x}_{i,j}(t) = -x_{i,j}(t) + \sum_{k,l=-N}^{N} (A_{k,l}(t)y_{i+k,j+l}(t) + B_{k,l}(t)u_{i+k,j+l}(t)) + I(t) \quad (1)$$

$$y_{i,j}(t) = f(x_{i,j}(t)) = 0.5 \times (|x_{i,j}(t)+1| - |x_{i,j}(t)-1|) \quad (2)$$

Where $1 \le i \le M$, $1 \le j \le N$, $A_{k,l}(t)$ is the feedback coefficient template, $B_{k,l}(t)$ is the input coefficient template, $I(t)$ is the bias, and $x_{i,j}(t)$, $y_{i+k,j+l}(t)$ and $u_{i+k,j+l}(t)$ are the state, output and input of the cell, respectively. Note that $A_{k,l}(t)$, $B_{k,l}(t)$ and $I(t)$ are time-variant templates, and t can be removed when time-invariant templates are used. For efficient implementation on a digital platform (e.g., CPU, GPU, FPGA), discrete approximation of CeNN is obtained by applying forward Euler approximation as shown in Equation (3), (4) and (5).

$$x_{i,j}(t) \cong (x_{i,j}(n+1) - x_{i,j}(n))/\Delta t \quad (3)$$

$$x_{i,j}(n+1) = x_{i,j}(n) + \Delta t(-x_{i,j}(n) + I(n) + \sum_{k,l=-N}^{N} (A_{k,l}(n)y_{i+k,j+l}(n) + B_{k,l}(n)u_{i+k,j+l}(n))) \quad (4)$$

$$y_{i,j}(n) = f(x_{i,j}(n)) = 0.5 \times (|x_{i,j}(n)+1| - |x_{i,j}(n)-1|) \quad (5)$$

Though it is possible to implement the computing operation with about 39 times operations in each iteration, $x_{i,j}$,$u_{i,j}$ and $y_{i,j}$ require extra buffers to store and transfer for subsequent operation. One solution is Full signal range(FSR) model of CeNN, which is more efficient by eliminating the intermediate variable $x_{i,j}$. FSR model is detailed in [19], and the modified equations are given as follows:

$$y_{i,j}(n+1) = f(x_{i,j}(n) + \sum_{k,l=-N}^{N} A_{k,l}(n) y_{i+k,j+l}(n) + w_{ij}) \tag{6}$$

with the offset term

$$w_{i,j} = \sum_{k,l=-N}^{N} B_{k,l} u_{i+k,j+l} + I \tag{7}$$

As for hardware implementation, the conversion from $x_{i,j}(n)$ to $y_{i,j}(n)$ makes it no longer necessary to store $x_{ij}(n)$ in registers any more. In this way $y_{ij}(n)$ is limited between 1 and -1 and can be expressed in fixed bit width, which is usually 8 bits including a signal bit. Moreover, FSR model also simplifies the iteration operation. Outputs of $x_{ij}(n)$ and $w_{ij}(n)$ are used for the next iteration to calculating $y_{ij}(n+1)$ while discarding $x_{ij}(n+1)$.

For efficient hardware implementation, discrete-time FSR CeNN models are widely adopted in FPGA implementations [16][14] [19][20] [15]. In particular, the computation is regular and reproducible in each iteration. In order to reduce the computational complexity, (6) and (8) can be rewritten to (8) and (9). The computation flow is divided into two processes: A process and B process, each of which corresponds to an iteration unit. Obviously, A process only executes Equation (8) and requires double multiplication operations of B process. By exploiting the FSR model, $w_{i,j}$ only needs one calculation through all iterations. Consequently, the computation flow can be implemented as a fully pipelined architecture, as the pipeline depth is the total number of Euler iterations desired.

$$A\ process: w_{i,j} = \sum_{k,l=-N}^{N} B_{k,l} u_{i+k,j+l} + I_{i,j} \tag{8}$$

$$B\ process: y_{i,j}(1) = \sum_{k,l=-N}^{N} A_{k,l} y_{i+k,j+l}(0) + w_{i,j} \tag{9a}$$

$$B\ process: y_{i,j}(2) = \sum_{k,l=-N}^{N} A_{k,l} y_{i+k,j+l}(1) + w_{i,j} \tag{9b}$$

.....

$$B\ process: y_{i,j}(N) = \sum_{k,l=-N}^{N} A_{k,l} y_{i+k,j+l}(N-1) + w_{i,j} \tag{9c}$$

2.2 Roofline Model

Figure 1 shows the roofline model based on I/O bandwidth and computational performance. The performance is bounded by the I/O bandwidth (BW) and the computational roof (CR). GFLOPS stands for the number of floating-point operations per second. Operational intensity (OI) features the

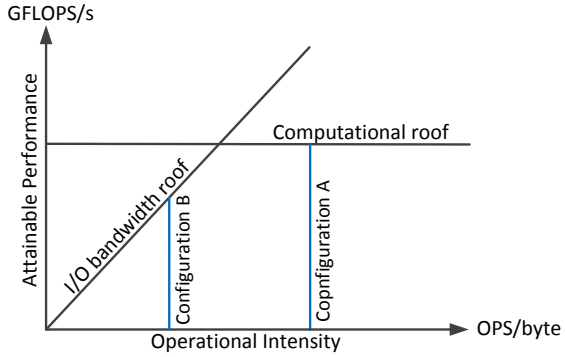

Figure 1: Illustration of the roofline model.

number of operations per byte memory accessed, and the product of OI and BW constitutes the I/O bandwidth roof. Generally BW and CR are considered as limits defined by the implementation details or obtained through benchmarks. Accordingly, the attainable performance is formulated as:

$$AP = min(CR, OI \times BW) \tag{10}$$

As depicted in Figure 1, OI is relatively high in Configuration A, the applications do not need to consume the entire I/O bandwidth, and the system are computation bound. As for Configuration B, BW becomes the limits, and the operation latency executed per byte are not enough to fit memory latency. CR is usually fixed for non-programmable hardware, while it is various for programmable FPGAs. Thus the sets located at the right side of or just on the roofline are supported by the platform. In roofline models for FPGAs, CR and OI are correlated. There exists a variety of configurations for a specific system, and the optimal configuration is determined by the system and the adopted hardware.

3 FRAMEWORK OVERVIEW

As shown in Figure 2, the designed architecture is composed of external memory, memory interface controller, on-chip input and output buffers, a computation acceleration unit (CAU), and AXI4 bus. The workload is divided in the temporal domain with a fully pipelined structure which is carried out by a chain of iteration units (IUs). Due to on-chip resource limitation, data are stored in external memory and cached in on-chip buffers before processed in CAU. CAU composes the vital part of the architecture, which is described in detail as follows:

- Iteration Unit A (IUA): The processor chain in CAU begins with IUA, which calculates both Equation (8) and (9a). As the template A_{ij} from Template RAM is constant in a time-invariant model, the result of Equation (8) remains the same in all iterations. Thus only one IUA is implemented at the very beginning. Input y of IUA can be ignored, and the first iteration feedback $y_{i,j}(0)$ is usually initialized as zero.
- Iteration Unit B (IUB): IUB indicates a number of identical modules, which form the second to the last

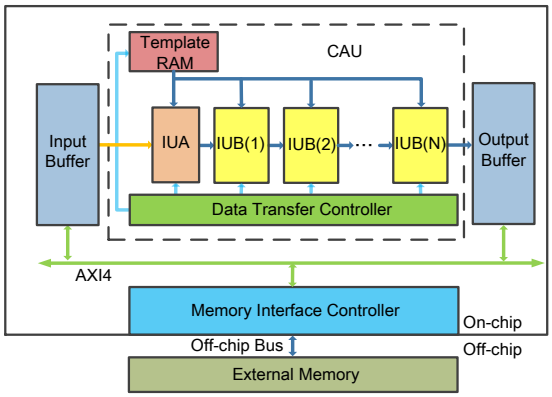

Figure 2: Overview of the proposed optimization framework.

unit in CAU. The first IUB, which is applied to calculate Equation (9b), captures $w_{i,j}$ and $y_{i,j}(1)$ from IUA. CeNN iterations are processed in a fully-pipelined architecture, i.e., the computation flow has to be divided into n+1 iterations in temporal domain before finally transferred into output buffer.

- Data Transfer Controller: Data transfer controller manages the data transmission among input buffer and output buffer, template RAM and FIFOs in pipelined iteration units.
- Template RAM: Template weights are space-invariant, which means that parameters are constant between consecutive images. Therefore, Template RAM needs only a few hardware resources for storage and memory access.

The following works are based on the optimization of CAU, which will be discussed in Section 4, 5 and 6.

4 SYSTEM LEVEL OPTIMIZATION

In this section, the parallel and tiling processing strategies and data-reuse techniques are proposed for an efficient parallel CeNN architecture.

4.1 Tiling Optimization

Parallel and tiling processing (PTP) is a sufficient approach correspondingly to make computation resources fully utilized and increase the maximum pixel rate. By separating workloads into N parallel arrays in spatial domain, N-PTP can be implemented in basic or advanced strategies, which allocate IUs in different structures resulting in different internal memory access operations. Basic strategy divides N parts of input images into N parallel arrays, which is composed of n FIFOs and processing elements (PEs). PE executes the computation operation of one pixel in one computation cycle. The topologies of IU is relatively easy to implement. However, the number of DRAM ports is the same as the tile size, which brings a challenge for limited hardware resources. As for advanced strategy, Data transfer controller split input images into separate data chunks efficiently. The capacity of data chunk is equal to tile size N, which has significantly different influence for the design. Data in each data chunk are arranged spatially with continuity and operated in one cycle. A data chunk occupies data by the least amount of DRAM ports when meeting the condition of maximum write/read bandwidth. Therefore, the number of DRAM ports decreases, which reduces hardware resources overheads.

Table 1: Data sharing relationships of matrix data (Sharing with the previous pixel (SWP), sharing with the latter pixel (SWL) and independent pixel (Independent)).

	Column 1	Column 2	Column 3
Pixels at TOL	Independent	SWP/SWL	SWL
Pixels at EOL	SWP	SWP/SWL	Independent
Other Pixels	SWP	SWP/SWL	SWL

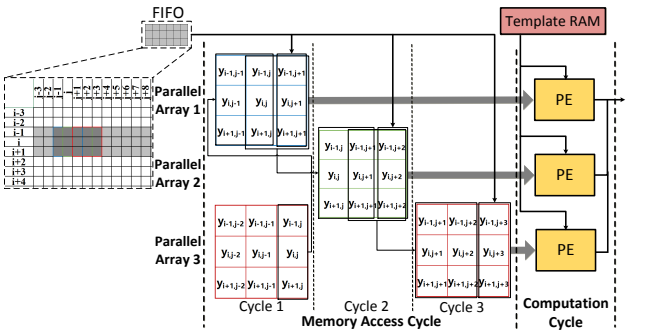

Figure 3: Data reuse optimization in advanced 3-PTP: Two columns of data are shared for a 3×3 template, resulting in 2/3 memory access reduction.

4.2 Data-reuse Optimization

As shown in Equation (3), the computation of each pixel is accompanied by two 3×3 matrices Y and U, which repeats in a regular manner in adjacent pixels. The optimizing solution is to minimize the memory access operation by utilizing data repetition. The distribution and location of repeated data determine the algorithm. As shown in TABLE 1, data sharing relationships depend on the location of pixels. All general pixels occupy the same data sharing relationships except for pixels at the top of line (TOL) and the end of line (EOL). Nine parameters of a matrix are divided into three groups by column, thus all pixels are categorized.

When calculations corresponding to Equation (8) and (9) are performed in IUs,data-reuse techniques eliminate excessive data read operation between IU and input buffer, which is a promising way to simplify memory access. As an example, the hardware implementation generated by a 3-PTP IU block design with data-reuse is shown in Figure 3. With accessing registers of adjacent parallel arrays instead of FIFOs, data-reuse optimization achieves 2/3 memory access reduction.

5 MODULE LEVEL OPTIMIZATION

The resource utilization of computation operation becomes a main constraints of system performance when the hardware implementation is applied. A large number of multiplications are required at the limits of embedded multipliers resources,

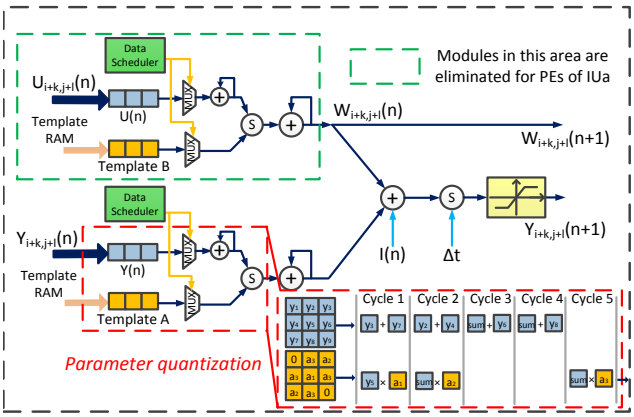

Figure 4: Quantization optimization for CeNN in PEs: Processing time is reduced from 9 cycles to 5 cycles, with repetition and sparsity optimization in Data Scheduler.

and data access latency limits the maximum computation rate. As for the module level, we present the further design exploration for IU to overcome these shortcomings.

Equation (9) depicts nine multiplications for CeNN calculation, which provides more opportunities to improve the computing process. As existing works have proved, 95%-100% of the embedded multiplier utilization in FPGA become a bottleneck in CeNN implementations [14]. Investigating the fact that embedded multipliers only occupy a small proportion of the core area, shifters are alternative methods for the integer operations. We propose quantization to optimize computation, which performs power-of-two conversion on all template parameters. This technique, although reduces little memory usage or computation complexity, has a significant impact on multiplier consumption by transferring multiplications into logic shifts.

Note that the template quantization brings two critical scenarios. First, with power-of-two parameter quantization, most of the templates occupy 5-6 repeated values, which enables repetition-induced optimization. Second, parameter quantization generates sparse matrix and prune the multiplications. The sparsity-induced and repetition-induced optimization are efficient in most CeNN applications, for the analysis of 87 tasks with 79 applications indicates that 78.2% templates are sparse matrix, while 94.4% templates contain repetitive parameters [9]. The quantization optimization is detailed in Figure 4.

Since memory access cycle is mostly one order of magnitude lower than the computation cycles, it has little impact for basic PTP strategy. However, memory access time for parallel arrays increases as well when applying advanced PTP strategy. Data transfer operations of FIFOs must be executed serially, and the proportion of computation time slumps. Thus, it is worthy to reorganize and manipulate the on-chip memory access and computation scheme to reduce processing latency. Investigating the memory access optimization in advanced PTP, we overlap parallel phases, and assign memory access cycles in a pipelined strategy.

6 DESIGN SPACE LEVEL OPTIMIZATION

Roofline model based on I/O bandwidth and computational performance is applied to evaluate AP of the target platform. The concept of byte-operation is adopted to analyze different kind integers exclusively, which defines a single operation of one byte integer. By classifying the operation complexity of different bit width, giga byte operations per second (GOPS/s) is used as the metric of CR.

We present computational roof of PE in IUA (CR_{PEA}) and IUB (CR_{PEB}) to indicate different structures in two kinds of IUs, which are used as basic elements to conclude CR_{FPGA}. As interpreted in Equation (11), *Parallel Size* presents the number of parallel PEs in one IU. Moreover, OI varies according to the configuration of PEs. Since the advanced PTP strategy reduces memory resource consumption and I/O ports usage, a significant improvement is achieved in Equation (12). Here $operations_{PE}$ indicates byte operations in one PE per computation cycle.

$$CR_{FPGA} = ((n_{IU} - 1) \times CR_{PEB} + CR_{PEA}) \times Parallel\ Size \tag{11}$$

$$OI = \begin{cases} operations_{PE} \times n_{IU} & \text{advanced PTP} \\ \frac{operations_{PE} \times n_{IU} \times \#Row}{\#Row + 2 \times (Parallel\ Size - 1)} & \text{basic PTP} \end{cases} \tag{12}$$

Attainable performance (AP) is determined by the minimum of I/O bandwidth (BW) roof and CR, which is depicted as follows:

$$AP = min(CR_{FPGA}, OI_{PTP} \times BW) \tag{13}$$

Figure 5 shows two roofline models with different I/O bandwidth applying all possible system level and module level optimization methods. There is a prerequisite that CR and BW are fixed for any applications in theoretical models. However, roofline models for FPGA-based frameworks are not so explicit, mainly resulted from the variable hardware resource overheads of memory interface controller. Therefore, We explore the design space for the platform and enumerate a set of implementations to select the target one (C) both satisfying CR and BW in Figure 5(b).

7 EXPERIMENT

We implemented the proposed CeNN framework on Xilinx Zybo board with Zynq XC7Z010 and Zedboard with Zynq XC7Z020. Implementation reports are generated to summarize the resource utilization and estimated latency accurately and efficiently.

Firstly, we proposed system and module level optimization as described above under the constraint of 3-PTP and 8 pipelines. Table 2 summarizes the performance comparison of the basic CeNN approach (Basic) [19] and the implementation of the proposed framework using system level optimization

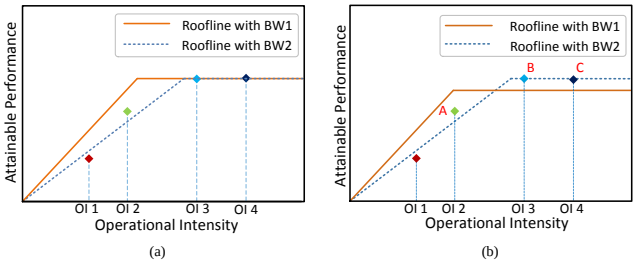

Figure 5: Comparison of (a) theoretical and (b) FPGA-based roofline models.

Table 2: Resource utilization and performance comparision.

	Basic	MLO	SLO	Basic	SLO+ MLO	SLO+ MLO+DSLO
Pipeline Depth	8	8	8	82	92	37
LUTs	15336	13632	10232	All	All	All
Computational Performance(Gops/s)	5.75	9.06	5.34	22.41	45.21	86.98
Computational Density (Gops/LUTs)	3.75× E-4	6.65× E-4	5.22× E-4	N/A	N/A	N/A
Improvement	1x	1.78x	1.39x	1x	2.01x	3.88x

(SLO) and module level optimization (MLO). For a fair comparison, we introduce the concept of computational density as a measure of energy efficiency which is hence independent of hardware platforms. In [21], computational density is defined as the average GOPS per area unit (GOPS/LUTs). As shown in the first three rows, MLO achieves a speedup of 1.58× and an energy efficiency improvement of 1.78× with the utilization of LUTs decreasing to 89% compared with the basic approach. SLO is 1.39× more energy efficient compared with the basic approach, while its resource utilization decreases to 66.72%.

For the purpose of analyzing the maximum attainable performance, we investigate the efficiency when combining different optimization techniques under various constraint scenarios. With design space level optimization (DSLO), resource utilization is constrained while numbers of PTP and pipeline depth is variable on the same FPGA platform. More experiments are shown in the last three columns of Table 2. The optimal model based on SLO+MLO+DSLO has the highest computational performance which achieves a 3.88x speedup. Moreover, a 3.54× energy efficiency enhancement can be roughly estimated by computational density compared with the basic model.

Implementations on different platforms are presented in Figure 6. Each point denotes the working mode that the FPGA may achieve, which depicts that 5-PTP tends to be an optimal model for Zybo, while 4-PTP for Zedboard. As OI increases, the attainable performance increases first until it comes to a local maximum where the peak performance is obtained. With overall optimization, the optimal solution can be selected for the given platform with desired performance and operational intensity.

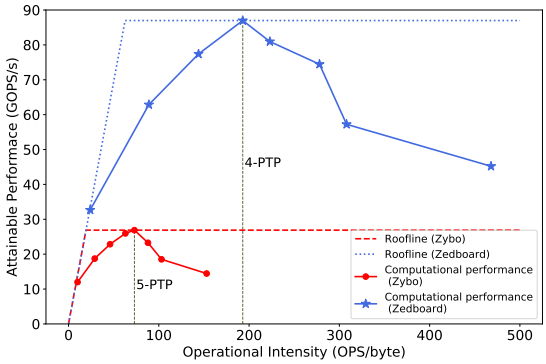

Figure 6: Roofline models of the proposed design space optimization on two different platforms (Zybo: 17600 LUTs, Zedboard: 53200 LUTs).

8 CONCLUSIONS

We propose an efficient multi-level optimized framework for the extended FPGA CeNN implementations. In system level, Parallel and tilling processing(PTP) and data-reuse optimization are applied. The computation flow is improved by separating the workload in spatial domain. In module level optimization, computation and memory access are optimized considering the limiting resources and memory bandwidth. In design space optimization, we apply the aforementioned optimization techniques of different CeNN scales in roofline models accordingly of attainable performance and operational intensity. Then the optimal CeNN solution can be selected with desired performance and operational intensity. Finally, evaluations of the proposed framework are presented on different FPGA platforms, and a speed up of 3.88× is achieved compared with existing implementations.

REFERENCES

[1] F. Al Machot, M. Ali, A. H. Mosa, C. Schwarzlmüller, M. Gutmann, and K. Kyamakya. Real-time raindrop detection based on cellular neural networks for adas. *Journal of Real-Time Image Processing*, pages 1–13, 2016.

[2] S. J. Carey, D. R. Barr, B. Wang, A. Lopich, and P. Dudek. Mixed signal simd processor array vision chip for real-time image processing. *Analog Integrated Circuits and Signal Processing*, 77(3):385–399, 2013.

[3] H.-C. Chen, Y.-C. Hung, C.-K. Chen, T.-L. Liao, and C.-K. Chen. Image-processing algorithms realized by discrete-time cellular neural networks and their circuit implementations. *Chaos, Solitons & Fractals*, 29(5):1100–1108, 2006.

[4] L. O. Chua and T. Roska. *Cellular neural networks and visual computing: foundations and applications*. Cambridge university press, 2002.

[5] M. Duraisamy and F. M. M. Jane. Cellular neural network based medical image segmentation using artificial bee colony algorithm. In *Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[6] O. B. Gazi, M. Belal, and H. Abdel-Galil. Edge detection in satellite image using cellular neural network. *system*, 8:9, 2014.

[7] H. Harrer, J. A. Nossek, T. Roska, and L. O. Chua. A current-mode dtcnn universal chip. In *Circuits and Systems, 1994. ISCAS'94., 1994 IEEE International Symposium on*, volume 4, pages 135–138. IEEE, 1994.

[8] J. Hills and Y. Zhong. Cellular neural network-based thermal modelling for real-time robotic path planning. *International Journal of Agile Systems and Management 20*, 7(3-4):261–281, 2014.

[9] K. Karacs, G. Cserey, Zarndy, P. Szolgay, C. Rekeczky, L. Kek, V. Szab, G. Pazienza, and T. Roska. Software library for cellular wave computing engines. *Cellular Sensory and Wave Computing Laboratory of the Computer and Automation Research Institute*, 2010.

[10] S. Lee, M. Kim, K. Kim, J.-Y. Kim, and H.-J. Yoo. 24-gops 4.5-mm^2 digital cellular neural network for rapid visual attention in an object-recognition soc. *IEEE transactions on neural networks*, 22(1):64–73, 2011.

[11] H. Li, X. Liao, C. Li, H. Huang, and C. Li. Edge detection of noisy images based on cellular neural networks. *Communications in Nonlinear Science and Numerical Simulation*, 16(9):3746–3759, 2011.

[12] D. Manatunga, H. Kim, and S. Mukhopadhyay. Sp-cnn: A scalable and programmable cnn-based accelerator. *IEEE Micro*, 35(5):42–50, 2015.

[13] G. Manganaro, P. Arena, and L. Fortuna. *Cellular neural networks: chaos, complexity and VLSI processing*, volume 1. Springer Science & Business Media, 2012.

[14] J. J. Martnez, J. Garrigs, J. Toledo, and J. M. Ferrndez. An efficient and expandable hardware implementation of multilayer cellular neural networks. *Neurocomputing*, 114:54–62, 2013.

[15] J. Muller, R. Wittig, J. Muller, and R. Tetzlaff. An improved cellular nonlinear network architecture for binary and greyscale image processing. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2016.

[16] R. Porter, J. Frigo, A. Conti, N. Harvey, G. Kenyon, and M. Gokhale. A reconfigurable computing framework for multi-scale cellular image processing. *Microprocessors and Microsystems*, 31(8):546–563, 2007.

[17] S. Potluri, A. Fasih, L. K. Vutukuru, F. Al Machot, and K. Kyamakya. Cnn based high performance computing for real time image processing on gpu. In *Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET), 2011 Joint 3rd Int'l Workshop on*, pages 1–7. IEEE, 2011.

[18] A. Rodrguez-Vzquez, G. Lin-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galn, F. Jimnez-Garrido, R. Domnguez-Castro, and S. E. Meana. Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(5):851–863, 2004.

[19] N. Yildiz, E. Cesur, K. Kayaer, V. Tavsanoglu, and M. Alpay. Architecture of a fully pipelined real-time cellular neural network emulator. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(1):130–138, 2015.

[20] N. Yildiz, E. Cesur, and V. Tavsanoglu. On the way to a third generation real-time cellular neural network processor. *CNNA 2016*, 2016.

[21] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 161–170. ACM, 2015.