DLAU: A Scalable Deep Learning Accelerator Unit on FPGA

Chao Wang, *Member, IEEE*, Lei Gong, Qi Yu, Xi Li, *Member, IEEE*,
Yuan Xie, *Fellow, IEEE*, and Xuehai Zhou, *Member, IEEE*

***Abstract*—As the emerging field of machine learning, deep learning shows excellent ability in solving complex learning problems. However, the size of the networks becomes increasingly large scale due to the demands of the practical applications, which poses significant challenge to construct a high performance implementations of deep learning neural networks. In order to improve the performance as well as to maintain the low power cost, in this paper we design deep learning accelerator unit (DLAU), which is a scalable accelerator architecture for large-scale deep learning networks using field-programmable gate array (FPGA) as the hardware prototype. The DLAU accelerator employs three pipelined processing units to improve the throughput and utilizes tile techniques to explore locality for deep learning applications. Experimental results on the state-of-the-art Xilinx FPGA board demonstrate that the DLAU accelerator is able to achieve up to 36.1× speedup comparing to the Intel Core2 processors, with the power consumption at 234 mW.**

***Index Terms*—Deep learning, field-programmable gate array (FPGA), hardware accelerator, neural network.**

I. Introduction

In the past few years, machine learning has become pervasive in various research fields and commercial applications, and achieved satisfactory products. The emergence of deep learning speeded up the development of machine learning and artificial intelligence. Consequently, deep learning has become a research hot spot in research organizations [1]. In general, deep learning uses a multilayer neural network model to extract high-level features which are a combination of low-level abstractions to find the distributed data features, in order to solve complex problems in machine learning. Currently, the most widely used neural models of deep learning are deep neural networks (DNNs) [2] and convolution neural networks (CNNs) [3], which have been proved to have excellent capability in solving picture recognition, voice recognition, and other complex machine learning tasks.

However, with the increasing accuracy requirements and complexity for the practical applications, the size of the neural networks becomes explosively large scale, such as the Baidu Brain with 100 billion neuronal connections, and the Google cat-recognizing system with one billion neuronal connections. The explosive volume of data makes the data centers quite power consuming. In particular, the electricity consumption of data centers in U.S. are projected to increase to roughly 140 billion kilowatt-hours annually by 2020 [4]. Therefore, it poses significant challenges to implement high performance deep learning networks with low power cost, especially for large-scale deep learning neural network models. So far, the state-of-the-art means for accelerating deep learning algorithms are field-programmable gate array (FPGA), application specific integrated circuit (ASIC), and graphic processing unit (GPU). Compared with GPU acceleration, hardware accelerators like FPGA and ASIC can achieve at least moderate performance with lower power consumption. However, both FPGA and ASIC have relatively limited computing resources, memory, and I/O bandwidths, therefore it is challenging to develop complex and massive DNNs using hardware accelerators. For ASIC, it has a longer development cycle and the flexibility is not satisfying. Chen *et al.* [6] presented a ubiquitous machine-learning hardware accelerator called DianNao, which initiated the field of deep learning processor. It opens a new paradigm to machine learning hardware accelerators focusing on neural networks. But DianNao is not implemented using reconfigurable hardware like FPGA, therefore it cannot adapt to different application demands. Currently, around FPGA acceleration researches, Ly and Chow [5] designed FPGA-based solutions to accelerate the restricted Boltzmann machine (RBM). They created dedicated hardware processing cores which are optimized for the RBM algorithm. Similarly, Kim *et al.* [7] also developed an FPGA-based accelerator for the RBM. They use multiple RBM processing modules in parallel, with each module responsible for a relatively small number of nodes. Other similar works also present FPGA-based neural network accelerators [9]. Yu *et al.* [8] presented an FPGA-based accelerator, but it cannot accommodate changing network size and network topologies. To sum up, these studies focus on implementing a particular deep learning algorithm efficiently, but how to increase the size of the neural networks with scalable and flexible hardware architecture has not been properly solved.

To tackle these problems, we present a scalable deep learning accelerator unit named DLAU to speed up the kernel computational parts of deep learning algorithms. In particular, we utilize the tile techniques, FIFO buffers, and pipelines to minimize memory transfer operations, and reuse the computing units to implement the large-size neural networks. This approach distinguishes itself from previous literatures with following contributions.

1) In order to explore the locality of the deep learning application, we employ tile techniques to partition the large scale input data. The DLAU architecture can be configured to operate different sizes of tile data to leverage the tradeoffs between speedup and hardware costs. Consequently, the FPGA-based accelerator is more scalable to accommodate different machine learning applications.
2) The DLAU accelerator is composed of three fully pipelined processing units, including tiled matrix multiplication unit (TMMU), part sum accumulation unit (PSAU), and activation function acceleration unit (AFAU). Different network

Manuscript received January 9, 2016; revised April 28, 2016; accepted June 13, 2016. Date of publication July 7, 2016; date of current version February 16, 2017. This work was supported in part by the NSFC under Grant 61379040 and Grant 61272131, in part by the Jiangsu Anhui Provincial Natural Science Foundation under Grant BK2012194, in part by the Anhui Provincial Natural Science Foundation under Grant 1608085QF12, in part by the CCF-Tencent Open Research Fund, in part by the CSC Fellowship, in part by the Open Project of State Key Laboratory of Computer ArchitectureICT-CAS, and in part by the Fundamental Research Funds for the Central Universities under Grant WK2150110003. This paper was recommended by Associate Editor D. Chen.

C. Wang, L. Gong, Q. Yu, X. Li, and X. Zhou are with the University of Science and Technology of China, Hefei 230027, China (e-mail: cswang@ustc.edu.cn; sa514002@mail.ustc.edu.cn; yuiq@mail.ustc.edu.cn; llxx@ustc.edu.cn; xhzhou@ustc.edu.cn).

Y. Xie is with the University of California at Santa Barbara, Santa Barbara, CA 93106 USA (e-mail: yuanxie@ece.ucsb.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2016.2587683

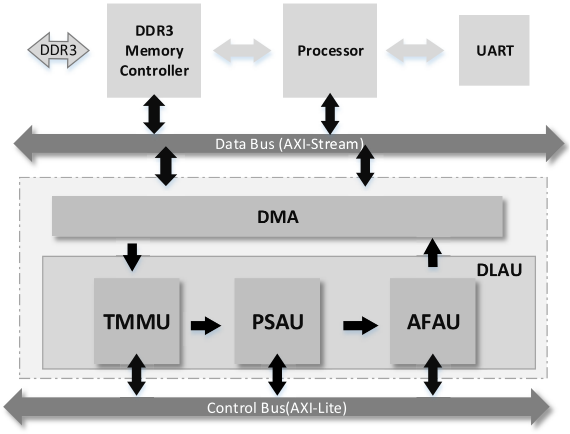

Fig. 1. DLAU accelerator architecture.

topologies such as CNN, DNN, or even emerging neural networks can be composed from these basic modules. Consequently, the scalability of FPGA-based accelerator is higher than ASIC-based accelerator.

II. Tile Techniques and Hot Spot Profiling

RBMs have been widely used to efficiently train each layer of a deep network. Normally, a DNN is composed of one input layer, several hidden layers and one classifier layer. The units in adjacent layers are all-to-all weighted connected. The prediction process contains feedforward computation from given input neurons to the output neurons with the current network configurations. Training process includes pretraining which locally tune the connection weights between the units in adjacent layers and global training which globally tune the connection weights with back propagation (BP) process.

The large-scale DNNs include iterative computations which have few conditional branch operations, therefore, they are suitable for parallel optimization in hardware. In this paper, we first explore the hot spot using the profiler. Results in Fig. 1 illustrates the percentage of running time including matrix multiplication (MM), activation, and vector operations. For the representative three key operations: 1) feed forward; 2) RBM; and 3) BP, MM play a significant role of the overall execution. In particular, it takes 98.6%, 98.2%, and 99.1% of the feed forward, RBM, and BP operations. In comparison, the activation function only takes 1.40%, 1.48%, and 0.42% of the three operations. Experimental results on profiling demonstrate that the design and implementation of MM accelerators is able to improve the overall speedup of the system significantly.

However, considerable memory bandwidth and computing resources are needed to support the parallel processing, consequently it poses a significant challenge to FPGA implementations compared with GPU and CPU optimization measures. In order to tackle the problem, in this paper we employ tile techniques to partition the massive input data set into tiled subsets. Each designed hardware accelerator is able to buffer the tiled subset of data for processing. In order to support the large-scale neural networks, the accelerator architecture are reused. Moreover, the data access for each tiled subset can run in parallel to the computation of the hardware accelerators.

TABLE I
Profiling of Hot Spots of DNN

Algorithms	Matrix Multiplication	Activation	Vector
Feedforward	98.60%	1.40%	
RBM	98.20%	1.48%	0.30%
BP	99.10%	0.42%	0.48%

Algorithm 1 Pseudocode Code of the Tiled Inputs

```
Require:
  Ni: the number of the input neurons
  No: the number of the output neurons
  Tile_Size: the tile size of the input data
  batchsize: the batch size of the input data
  for $n = 0; n < batchsize; n++$ do
    for $k = 0; k < Ni; k+= Tile\_Size$ do
      for $j = 0; j < No; j++$ do
        $y[n][j] = 0;$
        for $i = k; i < k + Tile\_Size \&\& i < Ni; i++$ do
          $y[n][j]+= w[i][j] * x[n][i]$
          if $i == Ni - 1$ then
            $y[n][j] = f(y[n][j]);$
          end if
        end for
      end for
    end for
  end for
```

In particular, for each iteration, output neurons are reused as the input neurons in next iteration. To generate the output neurons for each iteration, we need to multiply the input neurons by each column in weights matrix. As illustrated in Algorithm 1, the input data are partitioned into tiles and then multiplied by the corresponding weights. Thereafter the calculated part sum are accumulated to get the result. Besides the input/output neurons, we also divided the weight matrix into tiles corresponding to the tile size. As a consequence, the hardware cost of the accelerator only depends on the tile size, which saves significant number of hardware resources. The tiled technique is able to solve the problem by implementing large networks with limited hardware. Moreover, the pipelined hardware implementation is another advantage of FPGA technology compared to GPU architecture, which uses massive parallel SIMD architectures to improve the overall performance and throughput. According to the profiling results depicted in Table I, during the prediction process and the training process in deep learning algorithms, the common but important computational parts are MM and activation functions, consequently in this paper we implement the specialized accelerator to speed up the MM and activation functions.

III. DLAU Architecture and Execution Model

Fig. 1 describes the DLAU system architecture which contains an embedded processor, a DDR3 memory controller, a DMA module, and the DLAU accelerator. The embedded processor is responsible for providing programming interface to the users and communicating with DLAU via JTAG-UART. In particular it transfers the input data and the weight matrix to internal BRAM blocks, activates the DLAU accelerator, and returns the results to the user after execution. The DLAU is integrated as a standalone unit which is flexible and adaptive to accommodate different applications with configurations. The DLAU consists of three processing units organized in a pipeline manner: 1) TMMU; 2) PSAU; and 3) AFAU. For execution, DLAU reads the tiled data from the memory by DMA, computes with all the three processing units in turn, and then writes the results back to the memory.

In particular, the DLAU accelerator architecture has the following key features.

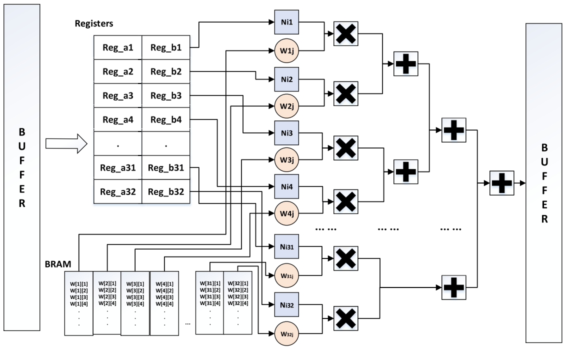
BUFFER
Registers
Reg_a1
Reg_b1
Reg_a2
Reg_b2
Reg_a3
Reg_b3
Reg_a4
Reg_b4
Reg_a31
Reg_b31
Reg_a32
Reg_b32
BRAM
W[1][1] W[1][2] W[1][3] W[1][4]
W[2][1] W[2][2] W[2][3] W[2][4]
W[3][1] W[3][2] W[3][3] W[3][4]
W[4][1] W[4][2] W[4][3] W[4][4]
W[31][1] W[31][2] W[31][3] W[31][4]
W[32][1] W[32][2] W[32][3] W[32][4]
Ni1
W1j
Ni2
W2j
Ni3
W3j
Ni4
W4j
Ni31
W31j
Ni32
W32j
BUFFER

Fig. 2. TMMU schematic.

FIFO Buffer: Each processing unit in DLAU has an input buffer and an output buffer to receive or send the data in FIFO. These buffers are employed to prevent the data loss caused by the inconsistent throughput between each processing unit.

Tiled Techniques: Different machine learning applications may require specific neural network sizes. The tile technique is employed to divide the large volume of data into small tiles that can be cached on chip, therefore the accelerator can be adopted to different neural network size. Consequently, the FPGA-based accelerator is more scalable to accommodate different machine learning applications.

Pipeline Accelerator: We use stream-like data passing mechanism (e.g., AXI-Stream for demonstration) to transfer data between the adjacent processing units, therefore, TMMU, PSAU, and AFAU can compute in streaming-like manner. Of these three computational modules, TMMU is the primary computational unit, which reads the total weights and tiled nodes data through DMA, performs the calculations, and then transfers the intermediate part sum results to PSAU. PSAU collects part sums and performs accumulation. When the accumulation is completed, results will be passed to AFAU. AFAU performs the activation function using piecewise linear interpolation methods. In the rest of this section, we will detail the implementation of these three processing units, respectively.

A. TMMU Architecture

TMMU is in charge of multiplication and accumulation operations. TMMU is specially designed to exploit the data locality of the weights and is responsible for calculating the part sums. TMMU employs an input FIFO buffer which receives the data transferred from DMA and an output FIFO buffer to send part sums to PSAU. Fig. 2 illustrates the TMMU schematic diagram, in which we set tile size = 32 as an example. TMMU first reads the weight matrix data from input buffer into different BRAMs in 32 by the row number of the weight matrix ($n = i\%32$ where n refers to the number of BRAM, and i is the row number of weight matrix). Then, TMMU begins to buffer the tiled node data. In the first time, TMMU reads the tiled 32 values to registers Reg_a and starts execution. In parallel to the computation at every cycle, TMMU reads the next node from input buffer and saves to the registers Reg_b. Consequently, the registers Reg_a and Reg_b can be used alternately.

For the calculation, we use pipelined binary adder tree structure to optimize the performance. As depicted in Fig. 2, the weight data and the node data are saved in BRAMs and registers. The pipeline takes advantage of time-sharing the coarse-grained accelerators. As a consequence, this implementation enables the TMMU unit to produce a part sum result every clock cycle.

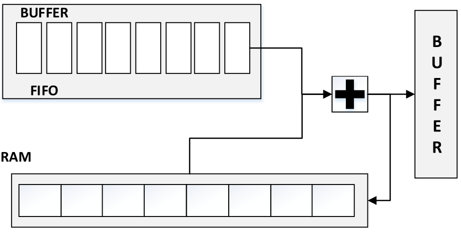

Fig. 3. PSAU schematic.

B. PSAU Architecture

PSAU is responsible for the accumulation operation. Fig. 3 presents the PSAU architecture, which accumulates the part sum produced by TMMU. If the part sum is the final result, PSAU will write the value to output buffer and send results to AFAU in a pipeline manner. PSAU can accumulate one part sum every clock cycle, therefore the throughput of PSAU accumulation matches the generation of the part sum in TMMU.

C. AFAU Architecture

Finally, AFAU implements the activation function using piecewise linear interpolation ($y = ai * x + bi, x \in [x_1, x_{i+1}]$). This method has been widely applied to implement activation functions with negligible accuracy loss when the interval between x$_i$ and x_{i+1} is insignificant. Equation (1) shows the implementation of sigmoid function. For $x > 8$ and $x \leq -8$, the results are sufficiently close to the bounds of 1 and 0, respectively. For the cases in $-8 < x \leq 0$ and $0 < x \leq 8$, different functions are configured. In total, we divide the sigmoid function into four segments

$$f(x) = \begin{cases} 0 & \text{if } x \leq -8 \\ 1 + a\left[\left\lfloor \frac{-x}{k} \right\rfloor\right]x - b\left[\left\lfloor \frac{-x}{k} \right\rfloor\right] & \text{if } -8 < x \leq 0 \\ a\left[\left\lfloor \frac{x}{k} \right\rfloor\right]x + \left[\left\lfloor \frac{x}{k} \right\rfloor\right] & \text{if } 0 < x \leq 8 \\ 1 & \text{if } x > 8. \end{cases} \quad (1)$$

Similar to PSAU, AFAU also has both input buffer and output buffer to maintain the throughput with other processing units. In particular, we use two separate BRAMs to store the values of a and b. The computation of AFAU is pipelined to operate sigmoid function every clock cycle. As a consequence, all the three processing units are fully pipelined to ensure the peak throughput of the DLAU accelerator architecture.

IV. Experiments and Data Analysis

In order to evaluate the performance and cost of the DLAU accelerator, we have implemented the hardware prototype on the Xilinx Zynq Zedboard development board, which equips ARM Cortex-A9 processors clocked at 667 MHz and programmable fabrics. For benchmarks, we use the Mnist data set to train the 784×M×N×10 DNNs in MATLAB, and use M×N layers' weights and nodes value for the input data of DLAU. For comparison, we use Intel Core2 processor clocked at 2.3 GHz as the baseline.

In the experiment we use tile size = 32 considering the hardware resources integrated in the Zedboard development board. The DLAU computes 32 hardware neurons with 32 weights every cycle. The clock of DLAU is 200 MHz (one cycle takes 5 ns). Three network sizes—64×64, 128×128, and 256×256 are tested.

TABLE II
COMPARISONS BETWEEN SIMILAR APPROACHES

Work	Network	Clock	Speedup	Baseline
Ly&Chow [5]	256×256	100MHz	32×	2.8GHz P4
Kim et.al [7]	256×256	200MHz	25×	2.4GHz Core2
DianNao [6]	General	0.98GHz	117.87×	2GHz SIMD
Zhang et.al [3]	256×256	100MHz	17.42×	2.2GHz Xeon
DLAU	256×256	200MHz	36.1×	2.3GHz Core2

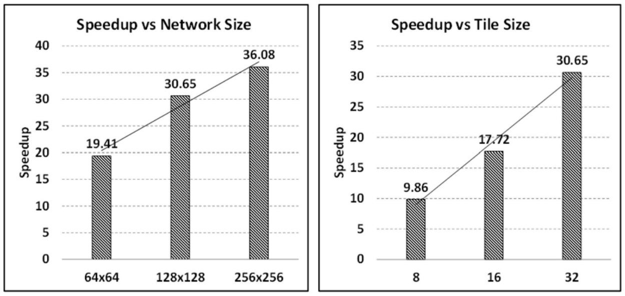

Fig. 4. Speedup at different network sizes and tile sizes.

TABLE III
RESOURCE UTILIZATION OF DLAU AT 32×32 TILE SIZE

Component	BRAMs	DSPs	FFs	LUTs
TMMU	32	158	25356	32461
PSAU	1	2	754	632
AFAU	2	7	2216	3291
Total	35	167	28326	36384
Available	280	220	106400	53200
Utilization	12.5%	75.9%	26.6%	68.4%

TABLE IV
RESOURCE COMPARISONS BETWEEN SIMILAR APPROACHES

Implementation	FPGA	BRAMs	DSPs	FFs	LUTs
Ly&Chow [5]	XC2VP70	257	N/A	30403	29885
Kim et.al [7]	N/A	589824	18	11790	7662
DLAU	XC7Z020	35	167	28326	36384

TABLE V
POWER CONSUMPTION OF THE UNITS

Component	Power	Component	Power
Accelerator-TMMU	189mW	Processor	1307mW
Accelerator-PSAU	5mW	DDR Controller	177mW
Accelerator-AFAU	25mW	Peripherals	26mW
Accelerator-DMA	15mW	Clocks	70mW
Accelerator-Total	234mW	System Total	1814mW

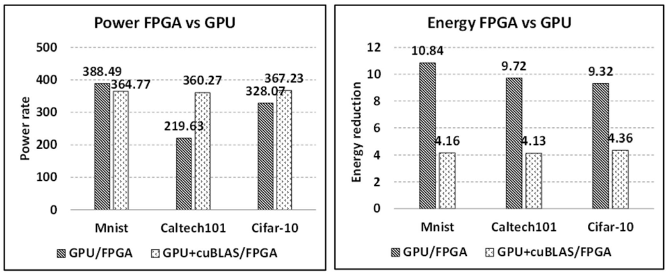

Fig. 5. Power and energy comparison between FPGA and GPU.

A. Speedup Analysis

We present the speedup of DLAU and some other similar implementations of the deep learning algorithms in Table II. Experimental results demonstrate that the DLAU is able to achieve up to 36.1× speedup at 256×256 network size. In comparison, Ly and Chow [5] and Kim *et al.* [7] presented the work only on RBM algorithms, while the DLAU is much more scalable and flexible. DianNao [6] reaches up to 117.87× speedup due to its high working frequency at 0.98 GHz. Moreover, as DianNao is hardwired instead of implemented on an FPGA platform, therefore it cannot efficiently adapt to different neural network sizes.

Fig. 4 illustrates the speedup of DLAU at different network sizes-64×64, 128×128, and 256×256, respectively. Experimental results demonstrate a reasonable ascendant speedup with the growth of neural networks sizes. In particular, the speedup increases from 19.2× in 64×64 network size to 36.1× at the 256×256 network size. The right part of Fig. 4 illustrates how the tile size has an impact on the performance of the DLAU. It can be acknowledged that bigger tile size means more number of neurons to be computed concurrently. At the network size of 128×128, the speedup is 9.2× when the tile size is 8. When the tile size increases to 32, the speedup reaches 30.5×. Experimental results demonstrate that the DLAU framework is configurable and scalable with different tile sizes. The speedup can be leveraged with hardware cost to achieve satisfying tradeoffs.

B. Resource Utilization and Power

Table III summarizes the resource utilization of DLAU in 32×32 tile size including the BRAM resources, DSPs, FFs, and LUTs. TMMU is much more complex than the rest two hardware modules therefore it consumes most hardware resources. Taking the limited number of hardware logic resources provided by Xilinx XC7Z020 FPGA chip, the overall utilization is reasonable. The DLAU utilizes 167 DSP blocks due to the use of the Floating-point addition and the Floating-point multiplication operations.

Table IV compares the resource utilization of DLAU with other two FPGA-based literatures. Experimental results depict that our DLAU accelerator occupies similar number of FFs and LUTs to Ly and Chow's work [5], while it only consumes 35/257 = 13.6% on the BRAMs. Comparing to the Kim *et al.*'s work [7], the BRAM utilization of DLAU is insignificant. This is due to the tile techniques so that large scale neural networks can be divided into small tiles, therefore, the scalability and flexibility of the architecture is significantly improved.

In order to evaluate the power consumption of accelerator, we use Xilinx Vivado tool set to achieve power cost of each processing unit in DLAU and the DMA module. The results in Table V depict that the total power of DLAU is only 234 mW, which is much lower than that of DianNao (485 mW). The results demonstrate that the DLAU is quite energy efficient as well as highly scalable compared to other accelerating techniques. To compare the energy and power between FPGA-based accelerator and GPU-based accelerators, we also implement a prototype using the state-of-the-art NVIDIA Tesla K40c as the baseline. K40c has 2880 stream cores working at peak frequency 875 MHz, and the max memory bandwidth is 288 (GB/s). In comparison, we only employ one DLAU on the FPGA board working at 100 MHz. In order to evaluate the speedup of the accelerators in a real deep learning applications, we use DNN to model three benchmarks, including Caltech101, Cifar-10, and MNIST, respectively. Fig. 5 illustrates the comparison between FPGA-based GPU+cuBLAS implementations. It reveals that the power consumption of GPU-based accelerator is 364 times higher

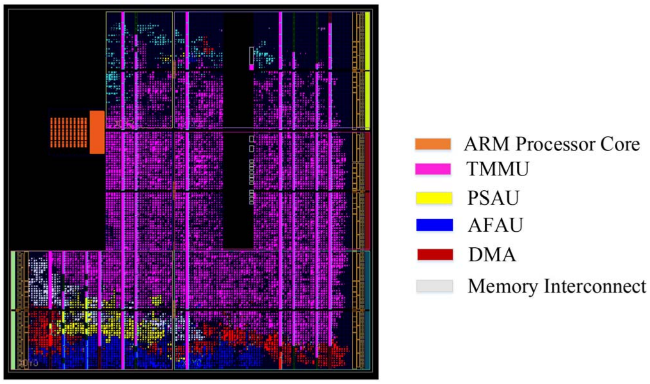

Fig. 6. Floorplan of the FPGA chip.

than FPGA-based accelerators. Regarding the total energy consumption, the FPGA-based accelerator is 10× more energy efficient than GPU, and 4.2× than GPU+cuBLAS optimizations.

Finally, Fig. 6 illustrates the floorplan of the FPGA chip. The left corner depicts the ARM processor which is hardwired in the FPGA chip. Other modules, including different components of the DLAU accelerator, the DMA, and memory interconnect, are presented in different colors. Regarding the programming logic devices, TMMU takes most of the areas as it utilizes a significant number of LUTs and FFs.

V. Conclusion

In this paper, we have presented DLAU, which is a scalable and flexible deep learning accelerator based on FPGA. The DLAU includes three pipelined processing units, which can be reused for large scale neural networks. DLAU uses tile techniques to partition the input node data into smaller sets and compute repeatedly by time-sharing the arithmetic logic. Experimental results on Xilinx FPGA prototype show that DLAU can achieve 36.1× speedup with reasonable hardware cost and low power utilization.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] J. Hauswald *et al.*, “DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers,” in *Proc. ISCA*, Portland, OR, USA, 2015, pp. 27–40.

[3] C. Zhang *et al.*, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proc. FPGA*, Monterey, CA, USA, 2015, pp. 161–170.

[4] P. Thibodeau. *Data Centers are the New Polluters*. Accessed on Apr. 4, 2016. [Online]. Available: http://www.computerworld.com/article/2598562/data-center/data-centers-are-the-new-polluters.html

[5] D. L. Ly and P. Chow, “A high-performance FPGA architecture for restricted Boltzmann machines,” in *Proc. FPGA*, Monterey, CA, USA, 2009, pp. 73–82.

[6] T. Chen *et al.*, “DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in *Proc. ASPLOS*, Salt Lake City, UT, USA, 2014, pp. 269–284.

[7] S. K. Kim, L. C. McAfee, P. L. McMahon, and K. Olukotun, “A highly scalable restricted Boltzmann machine FPGA implementation,” in *Proc. FPL*, Prague, Czech Republic, 2009, pp. 367–372.

[8] Q. Yu, C. Wang, X. Ma, X. Li, and X. Zhou, “A deep learning prediction process accelerator based FPGA,” in *Proc. CCGRID*, Shenzhen, China, 2015, pp. 1159–1162.

[9] J. Qiu *et al.*, “Going deeper with embedded FPGA platform for convolutional neural network,” in *Proc. FPGA*, Monterey, CA, USA, 2016, pp. 26–35.