Energy Efficient Techniques using FFT for Deep Convolutional Neural Networks

Nhan Nguyen-Thanh[1], Han Le-Duc[1], Duc-Tuyen Ta[1], and Van-Tam Nguyen[1,2]
[1] LTCI, CNRS, Télécom ParisTech, Université Paris Saclay, 75013, Paris, France
[2] Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA

Abstract*—Deep convolutional neural networks (CNNs) has been developed for a wide range of applications such as image recognition, nature language processing, etc. However, the deployment of deep CNNs in home and mobile devices remains challenging due to substantial requirements for computing resources and energy needed for the computation of high-dimensional convolutions. In this paper, we propose a novel approach designed to minimize energy consumption in the computation of convolutions in deep CNNs. The proposed solution includes (i) an optimal selection method for Fast Fourier Transform (FFT) configuration associated with splitting input feature maps, (ii) a reconfigurable hardware architecture for computing high-dimensional convolutions based on 2D-FFT, and (iii) an optimal pipeline data movement scheduling. The FFT size selecting method enables us to determine the optimal length of the split input for the lowest energy consumption. The hardware architecture contains a processing engine (PE) array, whose PEs are connected to form parallel flexible-length Radix-2 single-delay feedback lines, enabling the computation of variable-size 2D-FFT. The pipeline data movement scheduling optimizes the transition between row-wise FFT and column-wise FFT in a 2D-FFT process and minimizes the required data access for the element-wise accumulation across input channels. Using simulations, we demonstrated that the proposed framework improves the energy consumption by 89.7% in the inference case.

I. Introduction

Deep convolutional neural networks (CNNs) [1], [2] have achieved tremendous successes in a wide range of machine learning applications, including image recognition, computer vision, speech recognition, and natural language processing. Inspired by biological neural networks, deep CNNs includes one or more convolutional layers which act as linear filters to detect the specific features or patterns of the original data and offers significant improvements in performance over deep neural networks [2]. However, computations of convolutional layers requires substantial energy and time, preventing deployment of deep CNNs on home and mobile devices for future Internet of Thing (IoT) applications. For instance, AlexNet [2] a well-known CNN required 1.36×10^9 operations with 60 million parameters for backward and forward propagation through 5 layers of convolutions, resulting in several days for training by GPU [3]. To speed up computation of the CNN, several authors proposed to use Fast Fourier Transform (FFT)-based convolution method on GPU. FFT has several advantages as compared to other convolution methods, including rapid element-wise products and re-usability of transformed feature/filter maps [3]–[5]. However, the applications of FFT-based convolution still requires large computational resources and the feasibility of CNN on mobile and home devices remains unsolved.

In this paper, we propose a software-hardware architecture designed to minimize energy consumption required to support computations of high-dimensional convolutions of deep CNNs. The architecture consists of three major components: (i) an optimal selection method for FFT configuration associated with splitting input feature maps, (ii) a reconfigurable hardware architecture for computing high-dimensional convolutions based on 2D-FFT, and (iii) an optimal pipeline data movement scheduling. The architecture is based on three overarching design principles: divide-and-conquer, parallelization, and optimization. First, we leverage the overlap-and-save convolution-computing method [6] to split input feature maps by smaller size. This provides us the possibility of selecting the optimal FFT size that minimized the total energy consumption. Second, inspired by the direct convolution-based accelerator introduced in [7] for CNNs, we propose the use of a processing engine (PE) array, which enabled multiple parallel flexible-length Radix-2 single delay feedback(SDF) lines to be constructed for multple 1D-FFT calculations. The length of the 1D-FFT can be configured by choosing the number of PE stages and the suitable lengths of corresponding shift registers. Last, we use a feedback loop with a pipeline data movement scheduling to optimize the transition between row-wise FFT and column-wise FFT for 2D-FFT calculation. By this way, we can compute 2D-FFT without both extra matrix transpose memory and additional 1D-FFT processor. Furthermore, since the output of 2D-FFT is sequential, their element-wise product and accumulation across the input channels are conducted as soon as they appear. This helps to minimize data movement, consequently reduced energy consumption.

II. CNN background and FFT-based CNN

A. Convolution neural network

At a convolution layer, M images with K channels and size $H \times W$ are convoluted with F filters with K channels and size $R \times C$. Denote image elements by $I_{m,k,Sx+i,Sy+j}$ and filter elements by $G_{f,k,i,j}$. One element of convolution layer output is calculated by:

$$O_{m,f,x,y} = B_f + \sum_{k=1}^{K} \sum_{i=1}^{R} \sum_{j=1}^{C} I_{m,k,Sx+i,Sy+j} \times G_{f,k,i,j}, \\ 0 \le x \le X, 0 \le y \le Y, 0 \le f \le F, 0 \le m \le M \tag{1}$$

where $X = (W - C + S)/S$ and $Y = (H - R + S)/S$ with stride size S. B_f is the bias corresponding to the filter f. Without loss of generality, the output of a convolution layer is expressed by:

$$\mathbf{O}_{m,f} = \sum\nolimits_{k=1}^{K} \mathbf{I}_{m,k} \otimes \mathbf{G}_{f,k}, \tag{2}$$

where $\otimes$ represent 2D convolution operator.

B. FFT-based 2D convolution

Circular convolutions in spatial domain is computed by inversely transforming the element-wise products in Fourier domain, i.e., $\mathbf{I} \otimes \mathbf{G} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{I}) \odot \mathcal{F}(\mathbf{G}))$. In addition, using the linearity property of Fourier transform, Eqn. (2) is expressed by

$$\begin{aligned}\mathbf{O}_{m,f} &= \sum\nolimits_{k=1}^{K} \mathcal{F}^{-1}\left(\mathcal{F}(\mathbf{I}_{m,k}) \odot \mathcal{F}(\mathbf{G}_{f,k})\right) \\ &= \mathcal{F}^{-1}\left(\sum\nolimits_{k=1}^{K} \mathcal{F}(\mathbf{I}_{m,k}) \odot \mathcal{F}(\mathbf{G}_{f,k})\right).\end{aligned} \tag{3}$$

Obviously, the convolution based on Fourier Transform with 3D feature and filter maps provides not only the reuse element-wise products and transformed feature/filter maps but also low computation complexity for inverse Fourier transforms. The reason is that instead of computing the inverse Fourier transforms for each $(\mathcal{F}(\mathbf{I}) \odot \mathcal{F}(\mathbf{G}))$, only one inverse Fourier transform for the point-wise products sum is needed, see (3).

The FFT and the inverse FFT (IFFT) can be used to implement the convolution in Eqn. (3). Classically, to avoid aliasing of the cyclic convolution effect due to using FFT, we need to perform zero-padding both feature and filter maps to $(H + R - 1) \times (W + C - 1)$ 2D-FFT. The entire procedure of convolution computation using FFT is then done as follows. Step 1 - computes $(H + R - 1) \times (W + C - 1)$ 2D-FFT of both F filters and M image/feature maps with K channels. This requires $(M + F)K$ times of 2D-FFT with $(H + R - 1) \times (W + C - 1)$ points. Step 2 calculates $(H + R - 1) \times (W + C - 1)$ point-wise products of the corresponding filter and feature maps and accumulate the computed values across K channels. This step needs $MK\lceil(H + R - 1)/2\rceil(W + C - 1)$ complex multiplications and $M(K-1)(H + R - 1)(W + C - 1)$ complex additions. Step 3 computes IFFT of the accumulated maps to yield the convolution results. This step includes MF times of 2D-FFT with $(H + R - 1) \times (W + C - 1)$ points.

III. 2D-FFT Hardware architecture

A. Single-path Delay feedback 1D-FFT

An N-point Discrete Fourier Transform (DFT) of an input sequence $x[n]$ is defined as [6], [8]

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, k = 0, 1, \cdots, N - 1, \tag{4}$$

where $W_N = e^{-j2\pi/N}$ is the twiddle factor (phase factor or N^{th} primitive root of unity), and k is the frequency index. Direct implementation of (4) requires the order of operations $\mathcal{O}(N^2)$. In order to efficiently compute the DFT, FFT algorithms are used by decomposing the input sequence into smaller-size DFTs [8], which reduce the order of operations to $\mathcal{O}(N \log_2(N))$. In the high performance and real-time applications, widely used architectures for the FFT transform of length $N = r^v$, where r is called radix, are pipelined hardware architectures [9] since they provide high throughput and low latencies suitable for real time, as well as a reasonably low area and power consumption.

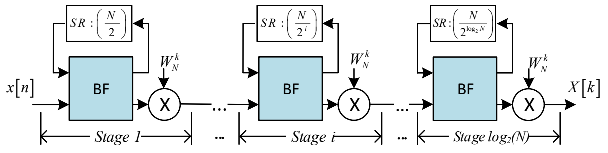
$SR:\left(\frac{N}{2}\right)$
$SR:\left(\frac{N}{2^i}\right)$
$SR:\left(\frac{N}{2^{\log_2 N}}\right)$
W_N^k
$x[n]$
BF
X
$X[k]$
Stage 1
Stage i
Stage log₂(N)

Figure 1. Hardware architecture of radix-2 Single-path Delay Feedback (SDF) Decimation In Frequency (DIF) pipeline FFT.

Fig. 1 shows a hardware architecture of the radix-2 Single-path Delay Feedback (SDF) DIF pipeline FFT [10]. This architecture consists of $\log_2(N)$ stages with $2log_2N$ complex adders and (log_2N-1) complex multipliers [11]. The detail of each stage is illustrated in Fig. 2. It encompasses three basic

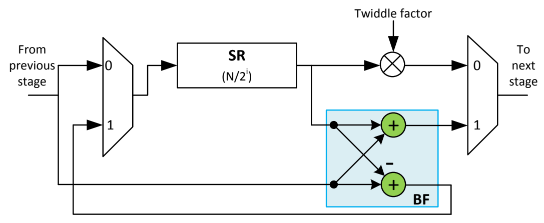

Figure 2. Detail architecture of the i^{th} stage in the radix-2 SDF FFT.

modules: *Butterfly (BF)*, *Shift Registers (SRs)* and *complex multiplier*.

Assuming that the input samples appear sequentially, $x(0), x(1)$, etc, the principle operations of each stage consists of three phases explained as follows. In the first phase, the first half $(N/2)$ of input samples are stored in the SRs. The SRs provide a time arrangement of samples for the corresponding stages. In the second phase, the reaming half of input samples are combined with the samples stored in SRs to execute the plus and minus operations in the BF module as

$$\left.\begin{aligned} s[i] &= x[i] + x\left[i + \tfrac{N}{2}\right] \\ s\left[i + \tfrac{N}{2}\right] &= x[i] - x\left[i + \tfrac{N}{2}\right] \end{aligned}\right., i = 0, 1, \ldots, \frac{N}{2}. \tag{5}$$

The adding result $s[i]$ is sent to the next stage without the twiddle factor multiplication meanwhile the result of difference $s[i + N/2]$ is eventually stored in the SRs. In the third phase, the results of difference previously stocked in the SRs will be multiplied with twiddle factors and then are sent to the next stage via a Multiplexer (MUX) as illustrated in Fig. 2. Note that at $(N+1)^{th}$ clock cycle, the first SR of the first BF stage is released and is re-used to store samples for column-wise 1D FFT computation described in Section III-B. The twiddle factors are pre-computed and stored in an array or Read-Only Memory (ROM). The complex multiplier with the twiddle factors is efficiently performed by COordinate Rotation DIgital Computer (CORDIC) algorithm. The MUX in this architecture is controlled by a controller and sends the sample values to the next stage in the specific clock cycle. The operation of the next stage is similar, except that the datum combination is executed by replacing $N/2$ with $N/4$ in (5) for BF computation. The entire procedure is completed after $(2N-1)$ clock cycles.

B. Hardware architecture of the 2D-FFT

A proposed hardware architecture of 2D FFT is shown in Fig. 3. There are two phases for the hardware architecture:

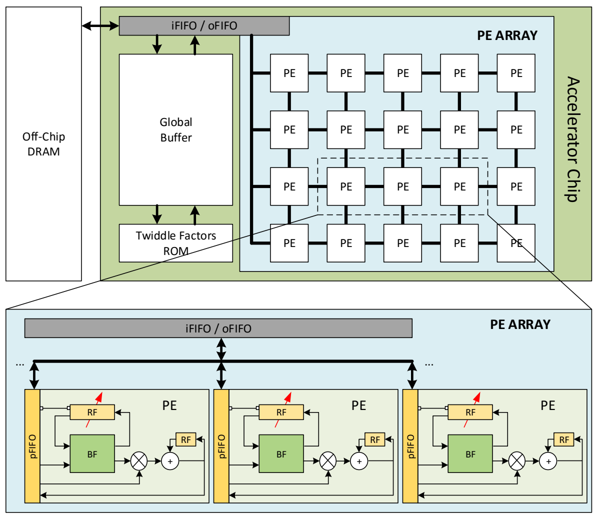

Figure 3. Hardware architecture of 2D FFT.

FFT phase and *element-wise accumulating phase*. In the FFT phase, the components required for a PE to compute the row-wise and column-wise 1D FFT are:

- 1 Register File (RF) which has a variable length and consists of SRs.
- 2 Adders with 16 bits precision-complex number for establishing a BF
- 1 multiplier with 16 bits precision-complex number for performing twiddle factors multiplication.

In element-wise accumulating phase, the components required for a PE are:

- 1 multiplier with 16 bits precision-complex number for implementing dot product
- 1 Adders and 1 RF for accumulating through input channels.

This hardware architecture contains a global buffer, iFIFO/oFIFO/pFIFO, ROM containing twiddle factors, and a PE array of a size $r \times c$. Each row of PE array is a flexible-length Radix-2 SDF line which is able to compute up to 2^c-point FFT. r rows working parallel enables to compute row-wise and column-wise 1D FFT, hence generating the 2D FFT results. By doing that, this architecture is formed by r parallel c-stage SDF lines. Note that the combination of rows and stages is flexible to construct many other sizes adapting several computational requirements. For an example, with an array of 16×10 PEs. One can construct 2×16 lines 5-stages SDF which allow us to compute 2D-FFT for input size of 32×32. But we can also establish 2 blocks of 16 lines 4-stages SDF which allow us to compute simultaneously two 2D-FFT for input size of 16×16. Similarly, we can select several other combinations.

To satisfy the requirement for both FFT and element-wise accumulation purposes, each PE is constructed as shown in Fig. 3. The multiplier can be used for multiplying either twiddle factor or coefficients from FFT-transformed filters/feature maps. The following adder and SR are used to perform accumulation through multiple channels for the convolution (with 4D) given in Eqn. (1).

IV. Data Flow and FFT Configuration Optimization

In general, the data flow dissipates much more energy in the hardware architecture. Therefore, the data flow optimization is needed, i.e., i) transition optimization between row-wise FFT and column-wise FFT in 2D-FFT process, ii) data access optimization for element-wise accumulation, and iii) optimal sizes of input segments which are associated with the optimal size of 2D-FFT.

A. Pipeline multiple-path delay feedback (MDF) 2D-FFT

As presented in Section III-A for the operation principle of the N-FFT radix-2 SDF line, the first output element appears at the clock N, and the first SR of the first BF stage is released at the clock $N+1$. Therefore, by inserting one clock delay, the SDF line can be reused for computing the next FFT step. In order to both transpose and feedback seamlessly, the following input rows must be delayed an amount which is equal to the order of the row, i.e., the row n is delayed n clocks. The addresses of both row and column of the final output are bit-reversed. However, due to the purpose of using FFT in which the 2D-FFT output is the intermediate for other computation processes, we do not need to implement bit-reversed permutation at this time. Instead, the permutation will be only performed before conducting IFFT.

Fig. 4 presents an example of executing 2D-FFT for an input matrix 4×4 by the proposed method. The hardware architecture includes 4 SDF lines containing two consequence BF stages and corresponding twiddle factor multipliers. The SR associated with the first BF stage includes 2 registers while the second one needs only 1 register.

B. Element-wise accumulation

Performing convolution based on FFT requires an element-wise multiplication step following the FFT step. Classically, this process is separately implemented after finishing FFT calculation. Fourier coefficients of image and filter are saved and reloaded to feed into multipliers. In order to reduce the energy for saving and reloading data, the multiplication is executed

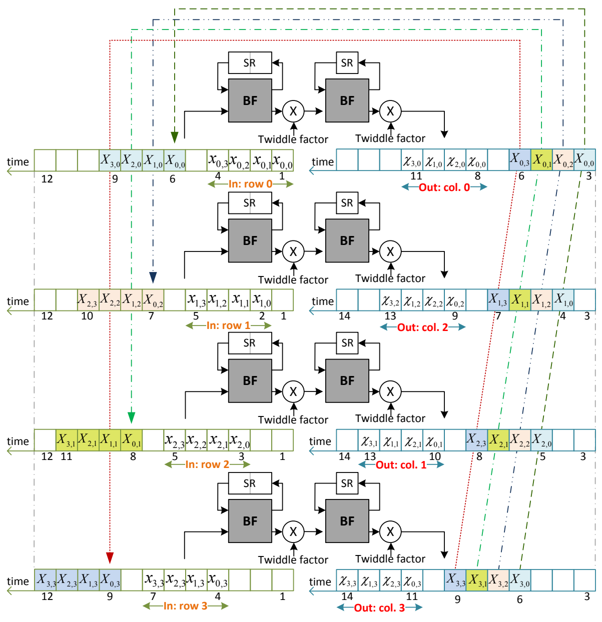

Figure 4. Data flow example of the 2D MDF FFT.

as soon as the sequences of final FFT coefficients appear at the output of SDF lines. The outputs of the multipliers are accumulated and kept in registers for implementing the convolution with 3D input images. The final FFT coefficients at the outputs of SDF lines are also be preserved for a future reuse. In that case, the input of the element-wise accumulator is loaded directly from memory and hence FFT computation energy is saved.

After completing the FFT coefficient product accumulation throughout input channels, the resulting elements are permuted based on bit-reversed address method before feeding back into SDF lines for computing IFFT to obtain the final convolution result.

C. Flexible-size FFT-based convolution

Using a fixed-size FFT hardware for computing the convolution requires a pre-padding process to adjust the image and filter size so that they are in accordance with the size of the hardware. This leads to redundancy in computation and data access. Therefore, the energy consumption for these process could be reduced by adopting a flexible-size FFT hardware. The flexibility of the hardware enables computing FFT with the size which is associated with the size of the input.

We leverage the extendable structure of SDF lines using Radix-2, which allows us to perform any 2^N-FFT by adding or removing the number of BF stages N, for establishing flexible size 2D-FFT. The flexible length is a power-of-2 number, i.e., 8,16,32,64,128, etc. When $H+R-1 < 2^{\lceil \log_2(H+R-1) \rceil}$, where H and R are the image and filter sizes, zero padding up to $2^{\lceil \log_2(H+R-1) \rceil}$ is needed.

Due to the limitation of hardware, the input size images can be larger than the maximum FFT length of hardware architecture. In that case, an overlap-and-save method for 1D convolution proposed in [6] is used for computing the 2D convolution. Fig. 5(a) and (b) show the operation principle of this method for 1D and 2D convolution, respectively. For 1D convolution, let us assume taking convolution between an $3H$-point input vector with a filter of length R using the FFT of length $(H+R-1)$. The input vector is split into three segments of length $H+R-1$ in which the first vector contains $(R-1)$ zero-padding points, and the second and the third vectors contain $(R-1)$ repeated points of the previous one. The three achieved vectors can be independently proceeded by FFT, element-wise multiplication, and IFFT to perform convolution. And the convolution is completed by removing $R-1$ premier points of each output vectors and merging the results. Analogously to 2D convolution, an original image is split into smaller partitions with the size approximating FFT size. The upper and left parts of a consecutive image partition are overlapped with the previous one or zero-padded in a roof tile pattern.

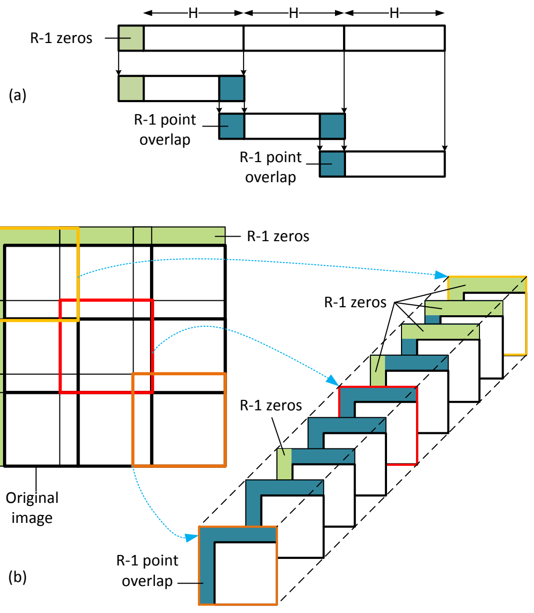

Figure 5. Overlapping split examples: (a) 1D and (b) 2D

D. FFT configuration optimization

To build up a tool for optimizing designed parameters for energy efficiency, we consider the computational complexity and the number data flow, which are the main consumption of the total energy. To simplify the analysis, hereafter, we assume that M feature maps and F filters are fed into the accelerator and the size of feature maps and the size of filters are $H \times H$ and $R \times R$, respectively. Both of them have K channels. In classical technique using the FFT in linear filtering in [6], the length of FFT is

$$L = 2^{\lceil \log_2(H+R-1) \rceil}, \tag{6}$$

that is sufficient to present all output images in frequency domain [6]. However, on the energy efficiency aspect, there

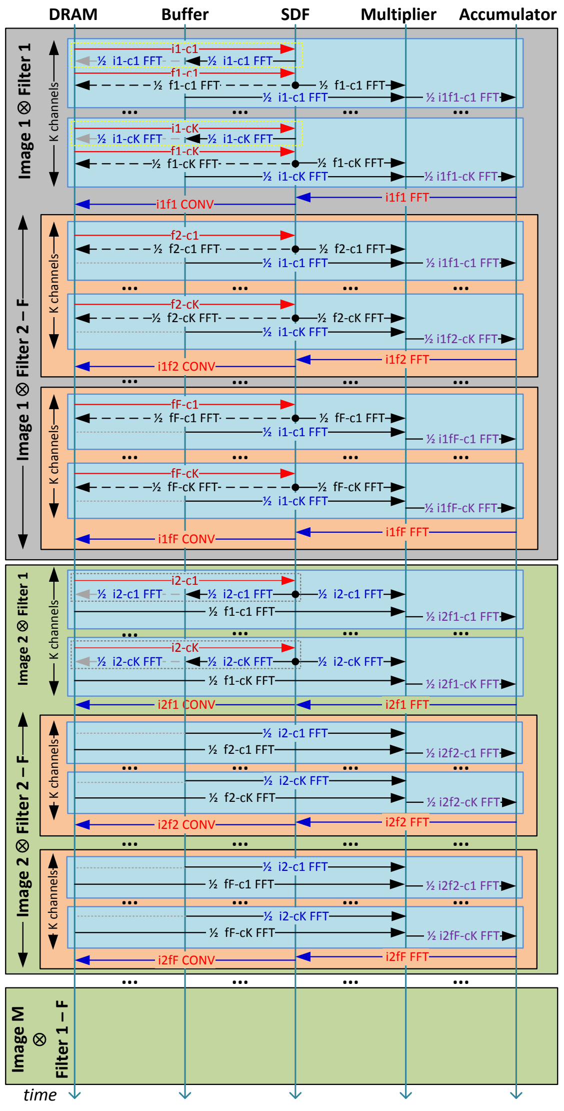

Figure 6. Data flow for computing the convolution of M images K channels and F filters K channels.

is no warranty that this FFT size is optimal. In practice, the hardware resource of the proposed flexible length radix-2 2D FFT is limited by CMOS technology. It means that the size of the input image is much larger than the 2D-FFT size in hardware. In such the case, we can divide the input image into tiles of size $(h+R-1)\times(h+R-1)$, where $(R-1)$ elements are overlapped with the neighbor tiles as shown in Fig. 5(b) and $h = L - R + 1$. This constructs $T = \lceil H/h \rceil^2$ tiles for each channel of input image. In order to achieve the minimum energy dissipation, a novel method splitting the input image into $L \times L$ input segments is proposed. In other words, the optimization problem to find the optimal value L is developed as follows.

The computation energy for computing 2D-FFT/IFFT and point-wise product are computed by:

$$E_{com}^{FFT} = E_{com}^{IFFT} = (2L\log_2 L)\, E^{(+)} + (L\log_2 L)\, E^{(\times)} \quad (7)$$

and

$$E^{\odot} = (L^2/2)E^{(\times)}, \quad (8)$$

where $E^{(+)}$ and $E^{(\times)}$ are the consumption energy of computing one complex addition and computing one complex multiplication, respectively. The total energy at SDF matrix is given by:

$$E^{SDF} = E_{com}^{FFT} + E_{mov}^{FFT}. \quad (9)$$

The energy for computing accumulation over K channel is computed by:

$$E_{com}^{Acc} = (K-1)\, L^2 E^{(+)}. \quad (10)$$

The energy for data movement in SDF pipeline 2D-FFT is calculated by:

$$E_{mov}^{FFT} = \left(L^2 \log_2 L\right) E^{RF} + L^2 E^{Array}, \quad (11)$$

where E^{RF} and E^{Array} are the consumption energy for one write and read RF and PE array. The energy for data movement in accumulator is given by:

$$E_{mov}^{Acc} = (K-1)\, L^2 E^{RF}. \quad (12)$$

Basing on the data flow in Fig. 6, the total energy consumption for implementing the convolution is computed by:

$$E = E_{image}^{FFT} MK + E_{filter}^{FFT} FK + (E_{\odot\&Acc} + E_{inv}) MF, \quad (13)$$

where

$$\begin{aligned} E_{image}^{FFT} &= H^2 E_{read}^{DRAM} + E^{SDF} + L^2 E_{write}^{Buffer}, \\ E_{filter}^{FFT} &= R^2 E_{read}^{DRAM} + E^{SDF} + E_{filter}^{Save\&Load}, \\ E_{inv} &= 2L^2 E^{Array} + E^{SDF} + S^2 E_{write}^{DRAM}, \\ E_{\odot\&Acc} &= K\left(L^2 E_{read}^{Buffer} + E^{\odot} + 2L^2 E_{write}^{RF}\right) \\ &\quad + (K-1)\, 2L^2 E^{(+)}, \end{aligned}$$

with $S = H - R + 1$, and

$$\begin{aligned} E_{filter}^{Save\&Load} &= 0, && M = 1 \\ E_{filter}^{Save\&Load} &= L^2 E_{write}^{DRAM} + (M-1)\, L^2 E_{read}^{DRAM}, && M > 1. \end{aligned}$$

The selection of FFT size is the solution of the following optimization problem:

$$\begin{aligned} \min_{i} \quad & E\left(M, H, F, R, K, L\right) \\ s.t. \quad & L = 2^i, L > R \end{aligned} \quad (14)$$

The optimal value of FFT size, i.e., solution of (14), can be easily found by a simple offline grid searching algorithm.

V. Simulation Results

In order to show the efficiency of the proposed method, we adopt the same normalized energy cost used in [7], i.e., $E^{DRAM} = 200\times, E^{Buffer} = 6\times, E^{Array} = 2\times$ and $E^{RF} = 1\times$. For the computational costs, the detailed values depend considerably on many technical aspects. Thus, without loss of generality, we assume that $E^{(+)} = 1\times$ and $E^{(\times)} = 2\times$. The parameters of convolutional layers using in our simulation is based on the configuration of those of AlexNet [2] given in Table I. The optimal FFT length for hardware (or optimal size

Table I
CONFIGURATION OF CONVOLUTIONAL LAYERS IN ALEXNET [2]

Layer	K	H	R	F
CONV1	3	227	11	55
CONV2	96	27	5	256
CONV3	256	13	3	384
CONV4	384	13	3	384
CONV5	384	13	3	256

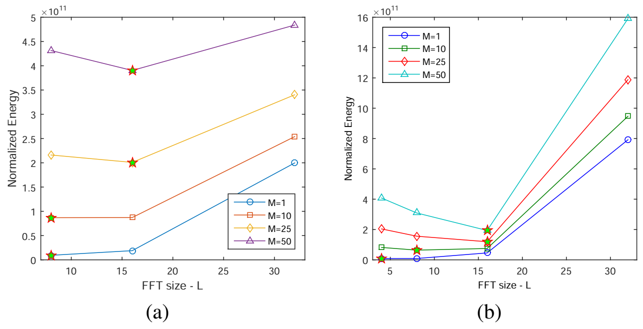

Figure 7. Optimal FFT sizes for layers of Alex Net (a-CONV2 and b-CONV3) with different values of input maps M

of input segments) is determined by minimizing the energy dissipation in Eqn. (14). Fig. 7 shows the estimated energy cost for executing CONV2 and CONV3 in AlexNet with four cases of the number of input feature maps ($M = 1, M = 10, M = 25$, and $M = 50$). We can observe that there is a certain optimal FFT size depending on the number of input feature maps. When the number of input features is small, the size of FFT should be also small. The reason is that using large FFT size require higher computational cost while the reusing times of transformed filters are small in this case. Therefore, there is a tradeoff between computational and reusing cost which leads to the existence of an optimal choice minimizing the total energy cost. Another interesting point is that for the considered cases, the optimal size of FFT is not large (≤ 16). The reason is due to the filter size which is not too large. Obviously, zero-padding the filter coefficients for large FFT can cause much more energy cost.

For a further view of evaluation, we compare the estimated energy cost in the cases of using our proposed optimal FFT size and the classical approach where the FFT size is determined by (6). It can be seen that, throughout the convolutional layers of AlexNet, our proposed method always has lower energy cost than the classical one. In total, the energy saving ratios are 40% and 89.7% for the cases of $M = 10$ and $M = 1$, respectively as shown in Fig. 8. Therefore, the proposed method is an efficient approach, especially for the inference case, where only a few of images are fed into.

VI. CONCLUSION

We have presented an integrated method for minimizing energy consumption in computing convolutions in deep CNNs. Firstly, we proposed a reconfigurable hardware architecture for computing high-dimensional convolutions based on 2D-FFT.

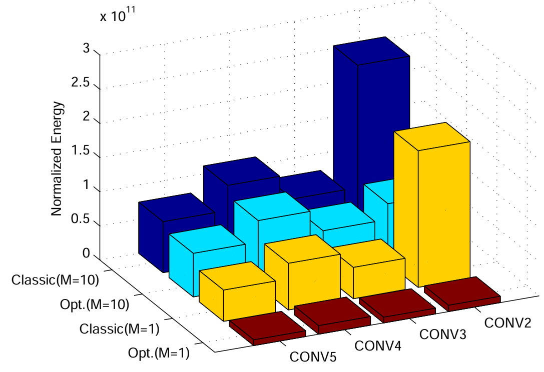

Figure 8. Energy comparison between the classical full-size FFT and the proposed methods

The hardware architecture contains a PE array, PEs of which are connected flexibly to form parallel flexible-length Radix-2 single delay feedback lines, enabling us to compute variable-size 2D-FFT. Secondly, we described an optimal pipeline data movement scheduling which optimizes the transition between row-wise FFT and column-wise FFT in a 2D-FFT process and minimizes the required data access for element-wise accumulation across input channels. Finally, we proposed an optimal selection method for FFT size associated with splitting input feature maps for minimizing energy consumption. The simulation results have shown a considerable improvement in energy consumption of the proposed architecture.

ACKNOWLEDGMENT

This research is funded by CORAMED project and CORTIF project

REFERENCES

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[3] A. Lavin, “Fast algorithms for convolutional neural networks,” *arXiv preprint arXiv:1509.09308*, 2015.
[4] M. Mathieu, M. Henaff, and Y. LeCun, “Fast training of convolutional networks through ffts,” *arXiv preprint arXiv:1312.5851*, 2013.
[5] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, “Fast convolutional nets with fbfft: A gpu performance evaluation,” *arXiv preprint arXiv:1412.7580*, 2014.
[6] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications.*, 4th ed. NJ, USA: Pearson, 2007.
[7] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, “Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks,” in *IEEE Int. Solid-State Circuits Conference, ISSCC 2016, Digest of Tech. Papers*, 2016, pp. 262–263.
[8] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1992.
[9] L. Rabiner and R. Schafer, *Theory and Applications of Digital Speech Processing*, 1st ed. NJ, USA: Prentice Hall, 2010.
[10] E. H. Wold and A. M. Despain, “Pipeline and parallel-pipeline fft processors for vlsi implementations,” *IEEE Transactions on Computers*, vol. C-33, no. 5, pp. 414–426, May 1984.
[11] T.-D. Chiueh, P.-Y. Tsai, and I.-W. Lai, *Baseband receiver design for wireless MIMO-OFDM communications.* John Wiley & Sons, 2012.