Hardware Acceleration Implementation of Three-Dimensional Convolutional Neural Network on Vector Digital Signal Processors

Weiwen Chen
National University of Defense Technology
College of Computer
Changsha, Hunan, China
e-mail: chenweiwen94@163.com

Yaohua Wang
National University of Defense Technology
College of Computer
Changsha, Hunan, China
e-mail: nudtyh@gmail.com

Chao Yang
Academy of Military Sciences
Beijing, China
e-mail: yc.nudt@gmail.com

Yong Li
National University of Defense Technology
College of Computer
Changsha, Hunan, China
e-mail: yongli@nudt.edu.cn

Abstract—**Convolutional neural networks (CNNs) have achieved great success in the field of computer vision. Researchers are currently focusing on more complicated three-dimensional (3D) CNNs and applying them to video processing. However, 3D CNNs have a huge number of calculations and parameters, which must be operated on an efficient computing platform. Digital signal processors (DSPs) are widely used in mobile terminals, high-performance workstations, and supercomputers. As a programmable computing platform, the DSP contains numerous multiplier-accumulator units (MACs) and supports vectorized calculation methods, which is very conducive to the efficient implementation of CNNs. To this end, it is of great significance to study the implementation of CNNs on a DSP. However, the current research on CNN acceleration on a DSP is still insufficient. This paper uses the FT-Matrix as the target DSP platform, which is a vector single instruction/multiple data (SIMD) architecture, and proposes a complete 3D CNN vectorized mapping method on the FT-Matrix, a block access scheme, and a hybrid convolutional layer optimization mapping method based on a sliding window and 3D Winograd algorithm. Our method can also be extended to other DSP platforms. Experimental results show that the algorithm can be executed accurately and efficiently on the DSP.**

Keywords-three-dimensional convolutional neural network (3D CNN); digital signal processor (DSP); hardware acceleration; vectorization mapping; winograd algorithm

I. INTRODUCTION

In recent years, the convolutional neural network (CNN) has achieved great success in use in image classification [1], [2], natural language processing [3], and pattern recognition [4]. Recently, 3D CNN has received more attention, and it has shown excellent performance in the field of video processing. Kai Yu et al. first proposed 3D CNN in 2013 [5], which is a more suitable algorithm for human behavior recognition than 2D CNN because the calculation of a 2D network stays on a plane, whereas 3D CNN gains time dimension characteristics when analyzing video data. Since then, various 3D CNN network structures have been proposed, such as ShapeNet [6], ObjectNet3D [7], VoxNet [8], C3D [9], which are widely used in video classification [10], medical imaging segmentation [11], and point-cloud semantic analysis [12]. The article by [13] demonstrates that 3D CNNs have good prospects and are promising for future success as 2D CNNs in ImageNet.

The outstanding performance of the CNN is based on the huge number of calculations and parameters. In particular, the 3D CNN is more complicated and larger than the 2D CNN. The convolution calculation usually takes more than 90% of the calculation time. Therefore, the execution of the algorithm depends on a high-performance computing platform. In scenarios with relatively simple requirements, application-specific integrated circuits and fieldprogrammable gate arrays (FPGAs) [14], [15] are often used as dedicated accelerators for CNNs, such as Google's tensor processing unit (TPU) [16], Cambricon's DaDianNao [17], and other artificial intelligence chips. The graphics processing unit (GPU) [18] is widely used in CNN scientific research and industrial computing. Many servers equipped with GPUs will be used for CNN training. Some workstations, mobile terminals, and supercomputers [19] also need to support CNN algorithms for development. In addition, DSPs are often used as general accelerators on these platforms. The DSP not only can operate conventional digital signal processing algorithms, such as fast Fourier transform, filtering algorithms, and so on, but can also support CNNs. In fact, as a programmable high-performance processor, the DSP supports vectorized calculation methods and contains numerous multiplier-accumulator units (MACs), which are very conducive to parallel convolution calculations in CNNs and have the characteristics of high energy efficiency and low cost.

In the past, CNN acceleration research on the DSP was insufficient. Some researchers have only implemented singlelayer or simple 2D CNN algorithms. This article presents a more in-depth study of the mapping and optimization of CNNs on DSPs. We propose to implement a 3D CNN on the FT-Matrix to complete this work. The FT-Matrix is a generalpurpose DSP with a vector single instruction/multiple data (SIMD) architecture, including two cores, where the peak performance reaches 200 giga floating-point operations per second (GFLOPS), and full-chip power consumption is 9 to 15 W. Moreover, 2D CNNs, such as AlexNet and GoogleNet, have been implemented on FT-Matrix in [20].

We propose a complete 3D CNN vectorization mapping method and a block access scheme on the DSP. Afterward, we also suggest a hybrid convolutional layer optimization mapping method combining a sliding window and 3D Winograd algorithm [21], which effectively accelerates the calculation of the convolutional layer. Our solution uses technologies, such as single-broadcast multi-computing, vectorized programming, and data reuse, which can be extended to other DSP platforms. We assessed the performance of the algorithm on the FT-Matrix, and the experimental results illustrate that our method can efficiently implement a 3D CNN.

II. Related Work

In previous research, Texas Instruments (TI) [22] implemented a simple CNN with two convolutional layers for object detection on their C66x DSP. Cadence [23] implemented a CNN for traffic-signal recognition in its Vision P6 DSP but only announced the method for a single layer of AlexNet. In addition, CEVA [24] implemented a 2D CNN on their XM4 Vison DSP for real-time target recognition, showing that the DSP can combine scalar vector calculations, flexible floating-point arithmetic, and data multiplexing in vision processing and is characterized by high efficiency and low power consumption and by being light weight. On the FT-Matrix DSP [20], a variety of 2D CNN algorithms, such as AlexNet, GoogleNet, and Yolo, have been implemented, primarily using the sliding window method to achieve the calculation of the convolutional layer. The CNN acceleration on the above DSPs is primarily for 2D CNNs or a single layer, and no special optimization exists for the convolutional layer. Therefore, this article provides a complete and more complicated mapping and optimization method for the 3D CNN on the DSP, which is a groundbreaking and meaningful exploration.

The organizational structure of this article is as follows. Section 3 introduces the background knowledge of 3D CNNs and the architecture of the FT-Matrix. Section 4 introduces the complete 3D CNN mapping and optimization method on the FT-Matrix. Section 5 shows the experimental results, and then a performance analysis is conducted. Section 6 summarizes this article and presents future research directions.

III. Background

A. 3D Convolutional Neaural Networks

The network structure of 3D CNNs is similar to 2D CNNs and comprises convolutional, pooling, fully connected (FC), and activation layers. The main difference is that the 3D CNN is calculated in a cube, which makes it easier to capture features on the timeline, while the 2D CNN stays on the 2D plane. The feature maps and filters for each layer have become 3D; therefore, the number of calculations and parameters is much larger. More importantly, it makes the optimization of the convolutional layer considerably more complicated.

The convolutional layer is the most important hidden layer in CNNs, and its primary function is feature extraction. More convolutional layers result in more complicated extracted features. In a 3D CNN, assuming that the size of the output feature (OF) map in the convolutional layer is M ∗X ∗Y ∗Z and M is the number of channels, the calculation of a single point OF[m][x][y][z] can be expressed as follows:

$$\mathrm{OF}[m][x][y][z]=\sum_{n=0}^{N-1}\sum_{i=0}^{I-1}\sum_{j=0}^{J-1}\sum_{k=0}^{K-1}\mathrm{IF}[n][x+i][y+j][z+k] \\ *\mathrm{Filter}[m][n][i][j][k]+\mathrm{Bias}[m]. \quad (1)$$

In the calculation process, each I ∗ J ∗ K filter is convolved with the 3D input feature (IF) map. The convolution is obtained by multiplying the points at the corresponding positions and adding them. After the convolution calculation of each input channel is completed, all channels are accumulated again to obtain OF maps. A total of M ∗N sets of filters and N IF maps exist to obtain M OF maps.

The role of the pooling layer is to reduce the scale of the feature map passed to the next convolutional layer and to lower the computational complexity while retaining the predominant features of the previous convolutional layer. Pooling also has the effect of accelerating training and reducing overfitting. Common pooling operations include average pooling and maximum pooling. As for 3D pooling, the length, width, and depth are compressed.

In the FC layer, each node in the current layer is connected to all nodes in the previous layer, and the high-dimensional data from the convolutional layer are nonlinearly transformed to integrate the extracted features. The FC feature makes the parameters of the FC layer the largest, and this layer has high memory requirements.

Next, the activation layer increases the nonlinearity of the CNN and allows neural networks to be applied to many nonlinear models. It is usually connected behind the convolutional layer and the FC layer, which uses the activation function to transform the image from the upper layer. The rectified linear unit (RELU) is the most commonly used activation function, which is f(x) = max(0,x).

B. 3D Winograd Algotithm

The convolutional layers are the most important hidden layers in CNNs and consume the main portion of calculation time spent. Therefore, reducing the number of calculations for the convolutional layers can effectively improve the overall performance of the CNN. The Winograd fast convolution algorithm reduces the number of multiplication operations and total calculations in convolution through matrix transformation (i.e., replacing computationally expensive operations with less expensive operations to reduce the total calculation overhead). It is an effective method to accelerate convolution. The following briefly describes the Winograd algorithm with a 1D convolution.

For example, Conv(a,b) represents the input feature tile (I) of size a and filter (W) of size b for convolution. For Conv(4,3), I is [i1,i2,i3,i4], and W is [w1,w2,w3]. The direct calculation is shown in Eq. (2), and a total of six multiplication operations and four addition operations are required.

$$\text{Conv}(4,3)=\begin{bmatrix} i1 & i2 & i3 \\ i2 & i3 & i4 \end{bmatrix}\begin{bmatrix} w1 \\ w2 \\ w3 \end{bmatrix}=\begin{bmatrix} i1*w1+i2*w2+i3*w3 \\ i2*w1+i2*w2+i2*w3 \end{bmatrix} \quad (2)$$

The calculation using the Winograd algorithm is shown in Eq. (3). First, matrix transformation is performed on I and W. Moreover, AT, BT, and G in Eq. (4) are transformation matrices. The transformation matrices corresponding to different a and b are different. Here, we only consider situations of a = 4 and b = 3. In addition, indicates the element-wise product (i.e., the points at the same position of two matrices are correspondingly multiplied to obtain a new matrix). We call the result of the temp result (TR). Finally, we perform another matrix transformation on TR to obtain the convolutional result. Two points are noted here. First, W is usually known in advance; therefore, we can complete the matrix transformation of W offline so that it does not use up actual calculation time. Second, from the form of AT and BT, the transformation of I and TR can be completed using addition. Therefore, only four multiplication operations (element-wise products) and eight addition operations (four additions for the I and four additions for the TR) are required when using the Winograd algorithm, thus reducing the number of multiplication operations compared to the direct calculation.

$$\text{Conv}(4,3)=A^T[(B^T I)\odot(GW)], \quad (3)$$

$$A^T=\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix},\ B^T=\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix},$$

$$G=\begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}. \quad (4)$$

The Winograd algorithm can be extended to 3D convolution and has a more significant optimization effect. In addition, Conv(4^3,3^3) represents the matrices I of size 4*4*4 and W of size 3*3*3 for convolution calculation, and the result is a matrix of 2*2*2. The formal expression of the Winograd algorithm can be given as follows (5):

$$\text{Conv}(4^3,3^3)=\text{trans\_of}\,[\text{trans\_if}\,(I)\odot\text{trans\_w}\,(W)]. \quad (5)$$

Since matrix transformation in 3D convolution is challenging to directly represent via matrix multiplication, the corresponding transformation is represented using the trans_x() function. Lan et al. [18] gave the general form of matrix transformation in the 3D Winograd algorithm (i.e., decomposed into multiple 1D Winograd algorithms). The 3D Winograd algorithm uses the same transformation matrices AT, BT, and G as the 1D convolution. The matrix transformation is given in Fig. 1. Moreover, InMat represents the matrix to be transformed, which can be I, W, or TR (which is in a 3D situation). In addition, Trans represents the corresponding transformation matrix, namely BT, AT, or G. Finally, OutMat represents the corresponding transformed matrix.

```
Input Matrix: InMat[m][m][m]
Output Matrix: OutMat[n][n][n]
Temp Matrix: T1[m][n][m], T2[m][n][n]
for i in [0:m]
    for j in [0:m]
        T1[i][0:n][j]=Trans* InMat[i][0:m][j]
for i in [0:m]
    for j in [0:n]
        T2[i][j][0:n]=Trans* T1[i][j][0:m]
for i in [0:n]
    for j in [0:n]
        OutMat[0:n][i][j] =Trans* T1[0:m][i][j]
```

Figure 1. Matrix transformation of the feature map or filters.

Although the transformation shown in the figure is expressed by matrix multiplication, the transformations of I and TR are implemented using addition in actual operation. Each matrix multiplication in the loop in the figure requires four additions to achieve, so the transformation of I requires 4*4*4*3 = 192 additions, and the transformation of TR is 4*(4*4+4*2+2*2) = 112 additions. In addition, W is known to be transformable offline so that it does not increase the total number of calculations. Therefore, in the calculation of Conv(4^3,3^3) using the Winograd algorithm, 4*4*4 = 64 multiplication operations are all derived from the point multiplication, and 192 + 112 = 304 multiplication operations are all derived from the matrix transformation. Table I shows the comparison of the Winograd algorithm and the direct calculation in the 3D convolution. The fast algorithm not only greatly reduces the calculation numbers of multiplication operations but also reduces the total number of calculations.

TABLE I. COMPARISON OF CALCULATION TIMES FOR THE DIRECT CALCULATION AND WINOGRAD ALGORITHM

	Direct Calculation	Winograd Algorithm
Multiplication Operations	216	64
Addition Operations	208	304
Total Computation	424	368

C. FT-Matrix Architecture

The FT-Matrix architecture is a multi-core vector processor. Its peak performance is 200 GFLOPS, and the basic frequency is 1 GHz, with 40 nm technology. The full-chip power consumption is 9 to 15 W, and the data width is 16/32 bit. The architecture of FT-Matrix is shown in Fig. 2. It includes two DSP cores capable of parallel operation, an ARM CPU core, and I/O nodes. These form a ring interconnect structure. Each DSP core includes a vector processing unit (VPU), scaler processing unit, vector registers (VR), and scaler registers. Single-core on-chip memory is divided into vector memory (VM) and scaler memory. Data are exchanged with external memory DDR through direct memory access components.

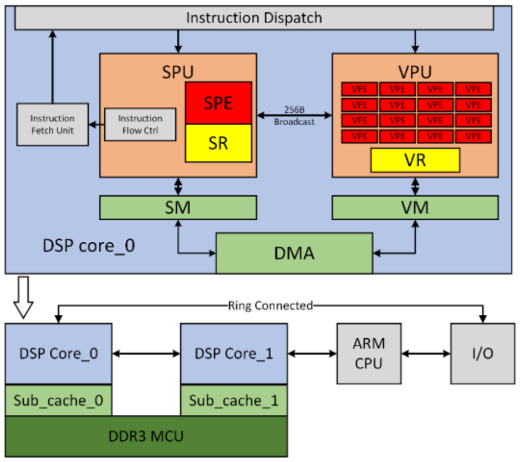

Figure 2. Architecture of the FT-Matrix.

The VPU contains 16 vector processing elements (VPEs), and each VPE has three MACs. The SIMD operation is supported (i.e., all VPEs in one core execute the same operation on different operands in one instruction). In addition to being able to perform scalar operations and the instruction stream, the scaler processing unit can also use broadcast instructions to transmit the same operands to multiple VPEs, which is a highly efficient data transmission method. The numerous MACs in the FT-Matrix, data broadcasting methods, and inter- and intra-core parallelism are keys to supporting the efficient execution of a CNN.

IV." Implementation

A. Complete 3D Convolutional Neural Network Mapping Method

This section uses the C3D [9] network as an example to illustrate the mapping method for 3D CNN on the DSP. The C3D network is a 3D CNN invented by Tran et al. in 2015, which is used for video classification. The recognition accuracy on the UCF101 video set can reach 85.2%, and the network is still improving. The input of the C3D is a series of continuous video frames of size 3*16*128*171, which must be pre-processed to a size of 3*16* 112*112, where three represents three color channels of red, blue, and green (RGB), and 16 is the number of consecutive video frames. Both the width and height of a single image are 112. The network structure of C3D is shown in theFig. 3. It consists of eight convolutional layers, five pooling layers, and three FC layers. The scale of the calculation results of each layer is also shown in Fig. 3.

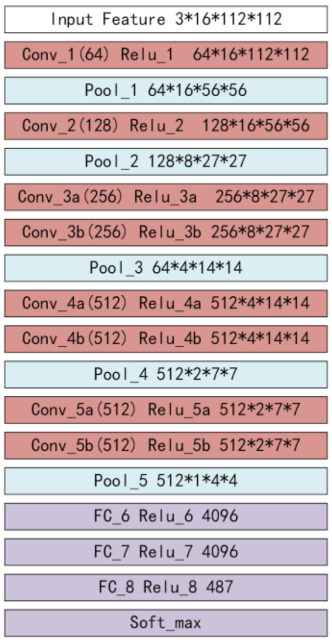
Input Feature 3*16*112*112
Conv_1(64) Relu_1 64*16*112*112
Pool_1 64*16*56*56
Conv_2(128) Relu_2 128*16*56*56
Pool_2 128*8*27*27
Conv_3a(256) Relu_3a 256*8*27*27
Conv_3b(256) Relu_3b 256*8*27*27
Pool_3 64*4*14*14
Conv_4a(512) Relu_4a 512*4*14*14
Conv_4b(512) Relu_4b 512*4*14*14
Pool_4 512*2*7*7
Conv_5a(512) Relu_5a 512*2*7*7
Conv_5b(512) Relu_5b 512*2*7*7
Pool_5 512*1*4*4
FC_6 Relu_6 4096
FC_7 Relu_7 4096
FC_8 Relu_8 487
Soft_max

Figure 3. C3D network.

We implemented C3D on the FT-Matrix by layers. We consider two important issues. The first is that many parallel multiplication and addition operations and comparisons exist in CNN calculations. To use the parallelism of CNN networks and take full advantage of DSP computing resources, we must properly vectorize these parallel operations based on the architecture of the FT-Matrix. The second issue is that 3D CNNs contain numerous weight parameters and temporary results. The parameters are moved as needed from DDR to the on-chip VM using direct memory access during the calculation. After that, the intermediate results must be moved from VM to DDR for the preparation of the next round of calculations. This is a very time-consuming process; therefore, both reducing the number of data moves and improving data utilization are crucial. Before start, we need to prepare. First, we extract the weight data from the binary file of the C3D pretrained model and store the data in the DDR of the DSP. Second, we must preprocess the video and reduce it to a size (3*16*112*112) that the network can receive. The storage order of the weights and images in DDR is according to the batch size, output channel, input channel, depth, height, and width. The data are calculated using a 32-bit floating point.

Regarding the convolutional layer, we first consider the vectorized mapping of the convolution calculation. As shown in Fig. 4, we provide the general form of 3D convolution, and the sizes of the IF map, weight, and OF map have been marked. Note that the vector instruction in the DSP requires the length of the operand to be 16. Seven convolutional layers exist in the C3D network, and the number of channels N (64, 128, 256, and 512) of the OF map are multiples of 16. Thus, we consider selecting points from different channels at the same position in the OF map to form a vector. Taking the first layer as an example, N = 64 = 16∗4, so four vectors can be composed, which are OF[0:15][i][j][k], OF[16:31][i][j][k], OF[32:47][i][j][k], and OF[48:63][i][j][k] because they perform the same calculation and do not interfere with each other. The complete pseudocode of the first convolutional layer is shown in the Fig. 5. We complete the convolution calculations using a sliding window. The weights and bias are first read into the VR as a vector. The IF maps are then broadcast to 16 VPEs and multiplied by the weight vectors. The intermediate result of the multiplication and addition operations is retained in the VR and accumulated with the next calculation result until the calculations for all channels are completed. The vectorized mapping of the subsequent convolutional layers is also similar to the first layer.

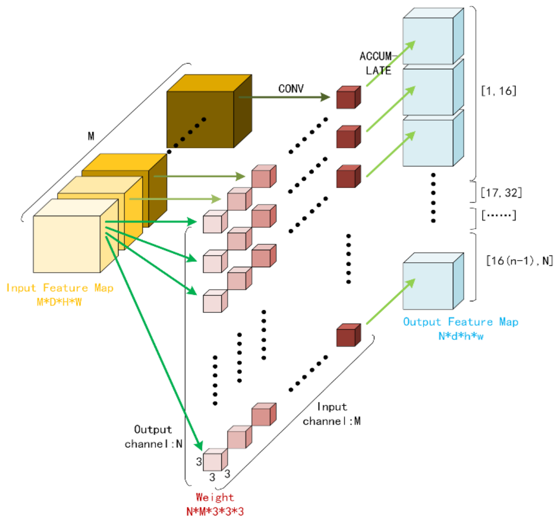

Figure 4. Vectorization mapping of the convolutional layer.

```
float          IF[3][18][114][114], img
vector_float   vOF[4][16][112][112], vB[4], vTmp[4]

for i = 0:16
  for j = 0:112
    for k = 0:112
      vTmp[0] = vB[0]
      ......
      vTmp[3] = vB[3]
      for r = 0:3
        for s = 0:3
          for t = 0:3
            for u = 0:3
              img = vec_broadcast(IF[r][i+s][j+t][k+u])
              vTmp[0] = vec_muladd(W[0][r][s][t][u], img, vTmp[0])
              ......
              vTmp[3] = vec_muladd(W[3][r][s][t][u], img, vTmp[3])
      vOF[0][i][j][k] = vTmp[0]
      ......
      vOF[3][i][j][k] = vTmp[3]
```

Figure 5. Pseudo code for convolutional layer implemented on DSP.

TABLE II. DATA VOLUME OF EACH CONVOLUTIONAL LAYER AND FULLY CONNECTED LAYER IN C3D

Layers	Weight Data(MB)	OF Data(MB)
CONV 1	0.02	49.0
CONV 2	0.85	24.5
CONV 3a + 3b	10.13	12.26
CONV 4a + 4b	40.5	3.06
CONV 5a + 5b	54	0.38
FC6 + FC7 + FC8	199.61	0.04

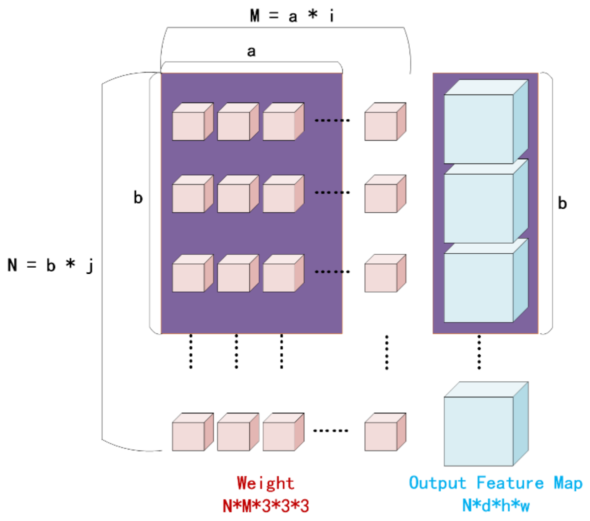

Figure 6. Partition for weight and output feature map.

The above algorithm simplifies the data access problem. The actual memory bandwidth of the DSP is about 12 GB/s. The time for the data transmission significantly affects the overall performance. In one convolution, the IF map is shared by the 16 VPEs and is directly transmitted from the DDR to the VPEs through a broadcast method. Therefore, the memory access cost of the IF maps is fixed. In contrast, the memory access of the OF maps and weight parameters is huge, as shown in Table II. Moreover, they must be moved from DDR to VM during the calculation. However, the VM is less than 800 KB, which is insufficient to load all the data required in one layer at a time (the on-chip memory cannot be designed to be too large or it affects the overall performance of the DSP). Due to the different amounts of weight and OF data in each layer, we partition the weight and OF separately. In OF maps, only the points at the same position between N channels are related to each other because they might be in a calculated result vector. Thus, the partition can be on d, h, and w, which does not increase the calculation time.

The weight partition is relatively complicated. As shown in Fig. 6, M ∗ N 3*3*3 filters (pink) and N OF maps (blue) are used in the calculation of a single layer. We assume M = a ∗ i and N = b ∗ j. In addition, we consider a ∗ b filters (purple area) to be a group, where a, b, i, j, M, and N are all positive integers, and N and b are multiples of 16. Then, $4a * b * \text{Mem(singlefilter)} + b * \text{Mem(singleoutputfeaturemap)} < \text{Mem(on-chip memory)}$ exists (i.e., the amount of data in a single group of weight and OF maps should be less than the VM capacity). Because the intermediate results, which are grouped by rows, are summed at the end, a is as large as possible to reduce the removal of the intermediate results during the accumulation process. The specific values of a and b in different layers should be determined according to the actual situation.

Other layers in the CNN take less than 10% of the total calculation time and are relatively simple to implement. Regarding the FC layer, the calculation volume is relatively small, but it has the largest number of weight parameters. Therefore, the memory bandwidth of the computing platform is a key factor affecting the performance of this layer. Moreover, C3D contains three FC layers, which can be regarded as a matrix and vector multiplication. Among them, FC6 is regarded as the multiplication of a 1 * 8192 vector and an 8192 * 4096 matrix. The situations for FC7 and FC8 are similar. The vector multiply accumulation can be implemented on the DSP. The weight data for the FC layers also must be transmitted and calculated by block like the convolutional layers.

Five pooling layers exist in the C3D network, and except for Pool_1, which has a filter and stride of 1*2*2 to avoid premature compression of the time dimension, all other pooling layers use a 2*2*2 filter and stride. The pooling layers in C3D use the max pooling function. In addition, 3D max pooling takes the maximum value in a given 2 * 2 * 2 matrix. We use the vector comparison units of the VPU in the DSP to complete the operation. The calculation of the pooling layer also requires vector mapping and data partitioning. Because no weight data exist, the situation is simpler than the convolutional layer. Only the division of the OF map must be considered, and it is not repeated here.

After each convolutional layer in C3D, an activation layer performs a y = max(x,0) function on each element of the OF map because the calculation of this layer is relatively simple. To reduce the data movement between DDR and VM, we directly merge it with the convolutional layer and use the vector comparison instruction to complete it.

B. *Optimization of Convolutional Layers*

In the C3D network, the filter size of all convolutional layers is 3*3*3. It has been demonstrated previously that, for the case of Conv(4^3,3^3), the Winograd algorithm can effectively reduce the multiplication operations and total number of calculations. We implemented the 3D Winograd algorithm on the DSP to accelerate the calculation of the convolutional layer. As shown in Fig. 7, we demonstrate the process of implementing Winograd in a complete 3D network. For the sake of clarity, we assume that the output channel N is 1 in this figure.

In Section 3.2, we discussed the case in which the IF map is 4*4*4 and the OF map is 2*2*2. In an actual 3D network, the IF map and OF map are much larger than this scale and are composed of multiple channels. Therefore, we must divide 1 the original IF map according to 4*4*4 with a stride of 2 to obtain 2 the regrouped IF map, and then perform the matrix transformation to obtain 3 the transformed IF map. When segmenting the IF map, if the boundary cannot be exactly divisible, we can add pad = 0 to the feature map, which does not affect the final result.

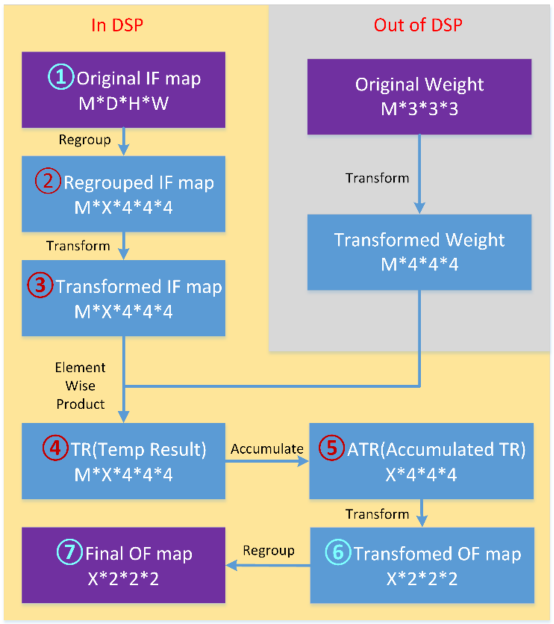
In DSP
Out of DSP
①Original IF map M*D*H*W
Regroup
②Regrouped IF map M*X*4*4*4
Transform
③Transformed IF map M*X*4*4*4
Element Wise Product
Original Weight M*3*3*3
Transform
Transformed Weight M*4*4*4
④TR(Temp Result) M*X*4*4*4
Accumulate
⑤ATR(Accumulated TR) X*4*4*4
Transform
⑥Transfomed OF map X*2*2*2
Regroup
⑦Final OF map X*2*2*2

Figure 7. Three-dimensional Winograd algorithm implementation process on digital signal processors.

Afterward, the wise-element product is performed with transformed filters to obtain 4. As shown in the gray part of the figure, to reduce the number of additions as much as possible, we completed the matrix transformation of the weight data in advance offline. This process changed all filters from 3*3*3 to 4*4*4 and increased the number of weight parameters; thus, the time for parameter transmission also increased. Subsequent processes are performed on the DSP. After the IF map of each channel and a set of filters are convolved, the result is finally accumulated to obtain an OF map. When using the Winograd algorithm, the 4*4*4 TR calculated from each input channel must also be accumulated.

However, because the matrix transformation of the TR is linear, we can take the accumulated TR to obtain 5 and then perform matrix transformation to obtain 6. This does not affect the calculation results but can effectively reduce the number of additions to convert multiple TR matrices. The acceleration effect depends on the current number of input channels of the convolutional layer. This optimization does not result from the Winograd algorithm but depends on the deployment by the specific network structure. After completing the element-wise multiplication, accumulation, and transformation, we concatenate all 2*2*2 OF maps to form 7, the final OF map. Throughout the process, the matrix transformation of the IF map and the result is performed using vector addition, whereas element-wise multiplication and accumulation are performed using vector multiply-add instructions.

V." Result Analysis

For direct calculation, we conducted performance tests on the FT-Matrix and on the Winograd optimization method. For a single layer, the total execution time for each method can be expressed using the following equations (6, 7):

$$T_d = 27MNdhw/P + (MDHW + Ndhw + 27MN)/B; \quad (6)$$

$$T_w = (8MNdhw + Trans_{IF,OF,W})/P + (17MDHW + 17Ndhw+64MN)/B; \quad (7)$$

where P and B respectively represent the actual performance and bandwidth of the chip. The total time comprises the calculation time and memory access time. The Winograd method has more calculations for the transformation matrix, and the volume of data for the feature maps and weight is larger. In addition, Fig. 8 lists the execution time for each convolutional layer in two ways. With the exception of Conv1, Conv5a, and Conv5b, the Winograd algorithm takes significantly less time than the direct calculation and can achieve up to 2.2 times the acceleration ratio, indicating that the Winograd algorithm can effectively accelerate the calculation of the convolutional layer.

In addition, we analyzed the reasons for the performance degradation of the first and last two layers. The execution time for each layer is primarily composed of I/O time and calculation time. The Winograd algorithm reduces the calculation time by reducing the total number of calculations, whereas the matrix conversion operations also increase the amount of data, making the I/O time longer. In the first layer, the size of the feature map is large. In the last two layers, the weight parameters are also large because the channels become larger. Thus, the reduction of the calculation time is insufficient to compensate for the increase in I/O time brought by the matrix conversion process, consequently making the total time longer. This also demonstrates that the acceleration performance of the Winograd algorithm is dependent on the storage bandwidth of the computing platform. In the end, we chose the hybrid calculation scheme to perform Winograd optimization on only the middle five convolutional layers.

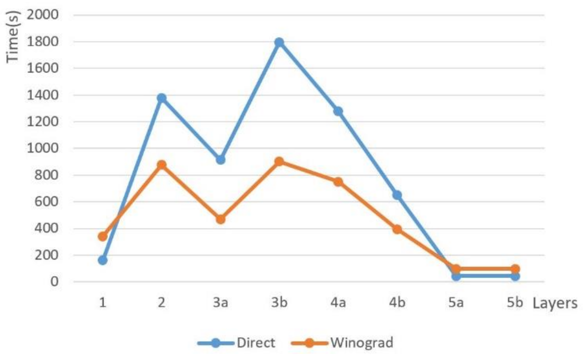

Figure 8. Three-dimensional Winograd algorithm implementation process on the digital signal processor.

VI. Conclusion and Future Work

This paper proposed a DSP-based 3D CNN hardware acceleration method. We implemented the direct calculation of the C3D algorithm on single-core mode and multi-core mode on DSPs of the FT-Matrix. The Winograd algorithm was then applied to optimize the mapping of the convolutional layer. We also compared its performance with the GPU. The experimental results demonstrate that the implementation of the C3D algorithm on the DSP has high computational efficiency and that it is an effective hardware acceleration method.

The Winograd algorithm can also be extended to the form of Conv(6^3,3^3), which reduces the number of calculations more effectively than Conv(4^3,3^3). In the next step, we plan to continue to study other forms of the Winograd algorithm to further improve the performance of the algorithm.

Acknowledgment

The authors thank Xiao Hu and Xianyou Sun for their contributions and helpful discussions regarding the development of this work.

References

[1] 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.

[2] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in 31st AAAI Conference on Artificial Intelligence, 2017.

[3] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587.

[5] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 1, pp. 221–231, 2012.

[6] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su et al., "Shapenet: An informationrich 3d model repository," arXiv preprint arXiv:1512.03012, 2015.

[7] Y. Xiang, W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese, "Objectnet3d: A large scale database for 3d object recognition," in European Conference on Computer Vision. Springer, 2016, pp. 160–176.

[8] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2015, pp. 922– 928.

[9] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 4489–4497.

[10] A. Diba, A. M. Pazandeh, and L. Van Gool, "Efficient two-stream motion and appearance 3d cnns for video classification," arXiv preprint arXiv:1608.08851, 2016.

[11] K. Kamnitsas, C. Ledig, V. F. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker, "Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation," Medical Image Analysis, vol. 36, pp. 61–78, 2017.

[12] F. Liu, S. Li, L. Zhang, C. Zhou, R. Ye, Y. Wang, and J. Lu, "3dcnndqn-rnn: A deep reinforcement learning framework for semantic parsing of large-scale 3d point clouds," in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5678-5687.

[13] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?" in Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2018, pp. 6546–6555.

[14]" J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song et al., “Going deeper with embedded fpga platform for convolutional neural network,” in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016, pp. 26–35.

[15]" J. Shen, Y. Huang, Z. Wang, Y. Qiao, M. Wen, and C. Zhang, “Towards a uniform template-based architecture for accelerating 2d and 3d cnns on fpga,” in Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2018, pp. 97–106.

[16]" N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers et al., “In-datacenter performance analysis of a tensor processing unit,” in 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2017, pp. 1–12.

[17]" Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun et al., “Dadiannao: A machine-learning supercomputer,” in Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE Computer Society, 2014, pp. 609–622.

[18]" Q. Lan, Z. Wang, M. Wen, C. Zhang, and Y. Wang, “High performance implementation of 3d convolutional neural networks on a gpu,” Computational Intelligence and Neuroscience, vol. 2017, 2017.

[19]" Y. Yu, J. Jiang, and X. Chi, “Using supercomputer to speed up neural network training,” in 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2016, pp. 942–947.

[20]" C. Yang, S. Chen, Y. Wang, and J. Zhang, “The evaluation of dcnn on vector-simd dsp,” IEEE Access, vol. 7, pp. 22301–22309, 2019.

[21]" A. Lavin and S. Gray, “Fast algorithms for convolutional neural networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4013–4021.

[22]" S. Jagannathan, M. Mody, and M. Mathew, “Optimizing convolutional neural network on dsp,” in 2016 IEEE International Conference on Consumer Electronics (ICCE). IEEE, 2016, pp. 371–372.

[23]" G. Efland, S. Parikh, H. Sanghavi, and A. Farooqui, “High performance dsp for vision, imaging and neural networks.” in Hot Chips Symposium, 2016, pp. 1–30.

[24]" Y. Siegel, “The path to embedded vision & ai using a low power vision dsp,” in 2016 IEEE Hot Chips 28 Symposium (HCS). IEEE, 2016, pp. 1–28.