Efficient Processing of Convolutional Neural Networks on SW26010

Yi Zhang, Bing Shu, Yan Yin, Yawei Zhou, Shaodi Li, Junmin Wu

Correction to: Chapter “Efficient Processing of Convolutional Neural Networks on SW26010” in: X. Tang et al. (Eds.): *Network and Parallel Computing*, LNCS 11783, https://doi.org/10.1007/978-3-030-30709-7_26

In the originally published version of this chapter, in section 2.2 and 3.3, in the second to last sentence “swDGEMM” was corrected to “swDNN”. Furthermore, in the last sentence of 3.3, “16” was corrected to “17”, and a reference to https://github.com/feifeibear/swDNNv1.0 has been added.

Efficient Processing of Convolutional Neural Networks on SW26010⋆

Yi Zhang[1](✉), Bing Shu[2], Yan Yin[1], Yawei Zhou[1], Shaodi Li[1], and Junmin Wu[1]

[1] University of Science and Technology of China, Hefei, Anhui, China
colezhan@mail.ustc.edu.cn
[2] Jiangnan Institute of Computing Technology, Wuxi, Jiangsu, China

Abstract. Artificial intelligence has developed rapidly in recent years. Deep neural networks are the basis of many artificial intelligence applications. How to accelerate the computational processing of deep neural networks is very important. To explor the potential for accelerating the process deep neural networks on various hardware platforms, we propose a convolutional neural network optimization method based on the Weight-Stationary for SW26010 processor. We re-circulate convolution loops and use hybrid DMA transmission mode to increase memory bandwidth and reduce memory access overhead. On top of those, further optimizations are done based on register communication, asynchronous DMA transfer double buffering, instruction scheduling and other schemes. Finally, we achieve a double-precision convolution performance over 2.4 Tflops, achieving 81% of the processor's peak performance. In multiple parameters, we achieve a proforamnce acceleration of $2.4-4.0\times$ speedup compared to the Tesla K80 GPU with cuDNNv7.

Keywords: SW26010 Processor · Convolutional Neural Networks · Weight-Stationary · Parallel Model · Many-core Architecture · Deep Learning.

1 Introduction

Deep neural networks (DNNs) are the foundation of many modern artificial intelligence (AI) applications. The high accuracy of DNNs is at the expense of high computational complexity and requires more computing power. In order to satisfy the computational requirements of DNNs, various acceleration hardware such as graphics processing units (GPU) [1] and FPGA [2] are applied to DNNs processing. Exploring DNNs calculation acceleration on various hardware platforms is a very valuable job. The SW26010 heterogeneous multicore processor is the processor chip of the Sunway TaihuLight supercomputer. In order to explore the combination of DNNs and SW26010, accelerate the processing of DNNs on SW26010, we first optimize the computational processing of the convolutional neural network (CNN), a common form of DNNs on SW26010, and

⋆ Supported by the National Key Research and Development Program of China under Grant 2018YFB1003601.

plan to expand to more forms in future. The main contributions of this work are as follows:

- We design a convolution algorithm based on the idea of Weight-Stationary and map it to SW26010.
- We analyze the DMA memory access characteristics of SW26010, use hybrid DMA transmission mode instead of bstride DMA transmission mode to increase memory bandwidth.
- We use an instruction scheduling method to reduce the idling time of computation units.
- We finally achieve over 2.4 Tflops double-precision convolution performance on SW26010, achieving 81% of the processor's peak performance.

2 Background and Related Work

2.1 Convolutional Neural Networks(CNN)

CNN is a common form of deep neural networks. In most CNNs, the operation of the convolutional layer occupies the largest portion of the total computation (more than 90%). The main operation of the convolutional layer is high-dimensional convolution. Let D be the input image, F be the convolutional filter, O be the output. Using the variable definitions in Fig 1, the algorithm formula for convolution operations [1] can be expressed as follows:

$$O[n,k,p,q] = \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} F[k,c,r,s] \cdot D[n,c,p+r,q+s] \tag{1}$$

2.2 Related Work

SW26010 Heterogeneous Many-core Processor is manufactured by the Shanghai High Performance IC Design Center through independent technology(see Fig. 2). As an emerging hardware platform, SW26010 has less work on efficient processing of DNNs. The authors of swDNN [3] have developed deep learning framework swCaffe [4] and deep learning acceleration library swDNN [3] for SW26010. However, swDNN does not consider the balance between memory access and computation, their double buffering scheme cannot completely cover the cost of memory access and their instruction scheduling scheme is not the best. In order to address these problems, we design a new algorithm based on the Weight-Stationary to balance the memory access and calculation, achieve almost complete cover-up of the memory access cost, and we also draw on the efficient instruction scheduling provided by swDGEMM. Finally we get very good performance by these optimization.

3 Optimization Of CNN On SW26010

3.1 Mapping CNN To SW26010

Considering the memory access characteristics of the SW26010 processor, the efficiency of accessing the main memory from the CPE using global read/store (gld/gst) is extremely low. The calculated data should be prefetched into the CPE's local memory (LDM) by means of DMA transfer when designing the algorithm. At the same time, considering the limited size of the LDM, it is necessary to keep the data reuse as much as possible when designing the algorithm.By analyzing the convolution algorithm, we can find that there are two forms of data reuse: one is the Output-Stationary, that is, output matrix is always kept in the LDM. When the matrix multiplication and addition operation associated with output matrix is completed, output matrix will be written back to the main memory. Another form of data reuse is the Weight-Stationary, which converts the loop's order, always keeps weight matrix in the LDM, reads in and reads out the output matrix in stages to complete the convolution calculation. We finally choose to design the algorithm based on the Weight-Stationary, it can reduce bandwidth requirements and adapt to the SW26010's limited bandwidth and memory. And we also use double buffering for the data transmission. The optimized algorithm is as algorithm 1.

Algorithm 1 CNN on SW26010 with double buffering

```
1: Load $O$ to the CPEs, $O$ are $bcp$ start matrixs(from$(0,0)$to$(bcp,0)$) of size $n \times k$
2: load $D$ to the CPEs, $D$ is $n \times c$ start matrix in$(0,0)$
3: for $cr = 0 : bcr : R$ do
4:    for $cs = 0 : S$ do
5:       Load $F$ to the CPEs,$F$ are $bcr$ matrixs(from$(cr,cs)$to$(cr+bcr,cs)$) of size $k \times c$
6:       for $cp = 0 : bcp : P$ do
7:          for $cq = 0 : Q$ do
8:             Load next $O$ to the CPEs for next computation, $O$ are $bcp$
               matrixs(from$(cp,cq)$to$(cp+2bcp,cq)$) of size $n \times k$
9:             for $ch = cp : cp + bcp + bcr - 1$ do
10:               Load next $D$ to the CPEs for next computation, $D$ is $n \times c$ matrix
                  in$(ch+1, cq+cs)$
11:               if $ch >= cp + cr$ and $ch < cp + cr + R$ then
12:                  $O_{cp,cq} += D_{ch,cq+S} \times F_{cr,cs}$
13:               end if
14:               sync
15:            end for
16:            Store current $O$ to the CPEs, $O$ are $bcp$ matrixs(from$(cp,cq)$to$(cp+$
               $bcp,cq)$) of size $n \times k$
17:            sync
18:         end for
19:      end for
20:   end for
21: end for
```

3.2 DMA Transfer Optimization

To map GEMM onto the processor array, we use matrix block multiplication to block the input and output data in an 8x8 array structure, to transfer data from main memory to LDM through the DMA, it is necessary to perform DMA stride transmission, resulting in low DMA transmission rate. But by the analysis of algorithm 1, we can find that the DMA stride transmission will be used to transfer the output matrix O back to the main memory only when the matrix multiplication and addition is completed, and the rest of the O's transmission can use a continuous DMA transmission mode. Therefore, we use a hybrid DMA transmission mode mixed with stride transmission and continuous transmission to improve the program's memory access bandwidth.

Para	Meaning
N	Number of images in mini-batch
C	Number of input feature maps
H	Height of input image
W	Width of input image
K	Number of output feature maps
R	Height of filter kernel
S	Width of filter kernel
P	Height of output feature maps
Q	Width of output feature maps

Fig. 1. Convolutional parameters

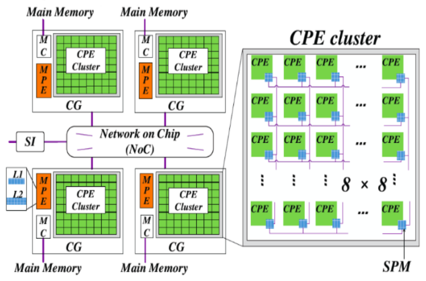

Fig. 2. The general architecture of the SW26010 processor

3.3 Instruction Scheduling

An important feature of the SW26010 is its dual instruction pipeline. Floating-point instructions and register communication instructions can be issued simultaneously by dual instruction pipeline. In some specific scenarios, great performance gains can be achieved by instruction scheduling, which was used in the design of swDNN [3] and swDGEMM [5]. After comparison, we selected the instruction scheduling provided by swDGEMM. Through the instruction scheduling method, the execution cycle of the instruction stream is reduced from 26 to 16, the overall operation efficiency of the program is greatly improved.

4 Results and Analysis

We use a number of sets of parameters to test our program performance and compare it to the K80 GPU using cuDNNv7. The final test results are shown in Fig. 3, Fig. 4, Fig. 5. Finally, we achieve a double-precision convolution performance over 2.4 Tflops, achieving 81% of the processor's peak performance, which is higher than swDNN [3]'s measured performance (only 54%) shown in their paper, and under different convolution filters, the performance is still stable(see Fig. 3). Compared to the K80 using cuDNNv7, we achieve $2.4 - 4.0\times$ acceleration.

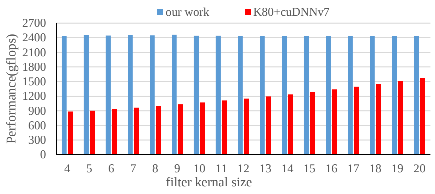

Fig. 3. Performance with different filter kernal (batchsize=128, output=64*64)

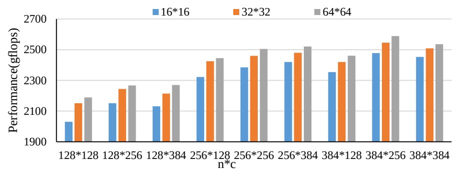

Fig. 4. Performance with different output size (batchsize=128, r=s=3)

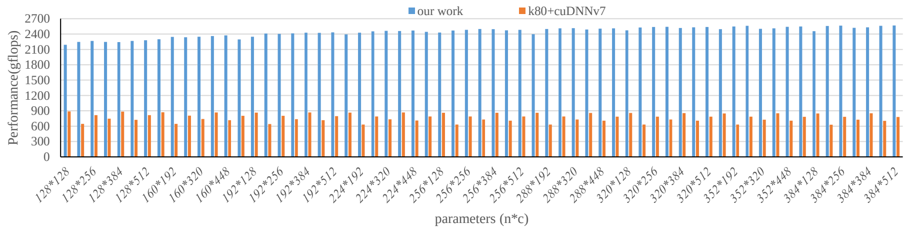

Fig. 5. Performance with different $n \times c$ (batchsize=128, output=64*64, r=s=3)

5 Conclusion

In this article, we present an efficient acceleration scheme for CNNs on SW26010 and finally achieve a double-precision convolution performance over 2.4 Tflops, achieving 81% of the processor's peak performance. In multiple parameters, we achieve a proforamnce acceleration of $2.4 - 4.0\times$ speedup compared to the Tesla K80 GPU with cuDNNv7.

References

1. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759 (2014) 1, 2.1
2. Chen, Y., Chen, T., Xu, Z., Sun, N., Temam, O.: Diannao family: energy-efficient hardware accelerators for machine learning. Communications of the ACM **59**(11), 105–112 (2016) 1
3. Fang, J., Fu, H., Zhao, W., Chen, B., Zheng, W., Yang, G.: swdnn: A library for accelerating deep learning applications on sunway taihulight. In: 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 615–624. IEEE (2017) 2.2, 3.3, 4
4. Li, L., Fang, J., Fu, H., Jiang, J., Zhao, W., He, C., You, X., Yang, G.: swcaffe: A parallel framework for accelerating deep learning applications on sunway taihulight. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER). pp. 413–422. IEEE (2018) 2.2
5. Jiang, L., Yang, C., Ao, Y., Yin, W., Ma, W., Sun, Q., Liu, F., Lin, R., Zhang, P.: Towards highly efficient dgemm on the emerging sw26010 many-core processor. In: 2017 46th International Conference on Parallel Processing (ICPP). pp. 422–431. IEEE (2017) 3.3