Available online at https://jcupt.bupt.edu.cn

The Journal of China Universities of Posts and Telecommunications

April 2024, 31(2): 105 – 112

Convolutional neural network adaptation and optimization method in SIMT computing mode

Feng Zhenfu(✉), Zhang Yaying, Yang Lele, Xing Lidong

School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, China

Abstract

For studying and optimizing the performance of general-purpose computing on graphics processing units (GPGPU) based on single instruction multiple threads (SIMT) processor about the neural network application, this work contributes a self-developed SIMT processor named Pomelo and correlated assembly program. The parallel mechanism of SIMT computing mode and self-developed Pomelo processor is briefly introduced. A common convolutional neural network (CNN) is built to verify the compatibility and functionality of the Pomelo processor. CNN computing flow with task level and hardware level optimization is adopted on the Pomelo processor. A specific algorithm for organizing a Z-shaped memory structure is developed, which addresses reducing memory access in mass data computing tasks. Performing the above-combined adaptation and optimization strategy, the experimental result demonstrates that reducing memory access in SIMT computing mode plays a crucial role in improving performance. A 6.52 times performance is achieved on the 4 processing elements case.

Keywords parallel computing, single instruction multiple threads (SIMT), convolutional neural network (CNN), memory optimization

1 Introduction

The continuous development and progress of artificial intelligence technology inject a powerful impetus into modern society. Neural network training requires large-scale data and sufficient computational power. In network training platforms, hardware accelerators such as GPGPU, field-programmable gate array (FPGA), and application specific integrated circuit (ASIC) are necessary. These years, GPGPUs are the first choice for running deep learning algorithms. NVIDIA provides a compute unified device architecture (CUDA) parallel computing framework[1-2] for running deep learning algorithms on GPGPUs. NVIDIA also proposed a parallel computing paradigm called SIMT to deploy efficiently algorithms on GPGPUs.

SIMT processor is an extended branch architecture of single instruction multiple data (SIMD) processor that adds the concept of warp on the basis of SIMD. Warp here borrows the concept from NVIDIA[2]. By allocating the data according to the threads and storing data as a private register unit, the parallel operation of the data can be realized under a single identical instruction, which is shown in Fig. 1. However, the registers owned by different threads in SIMT are private and cannot directly communicate with each other, threads can only communicate with each other through shared memory and synchronization mechanism.

Received date: 04-12-2023

Corresponding author: Feng Zhenfu, E-mail: fengzhenfu@xupt.edu.cn

DOI: 10.19682/j.cnki.1005-8885.2024.0008

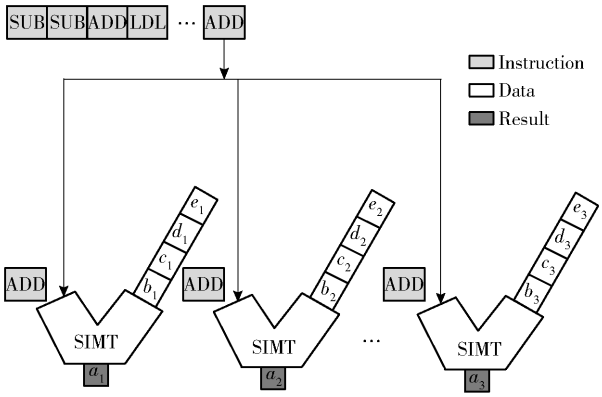

Fig. 1 SIMT working mechanism

SIMT architecture is similar to the vector-based architecture of SIMD and essentially belongs to the category of SIMD architecture. SIMT architecture is mainly designed to execute the parallel multiple threads and to realize the control of multiple processing elements (PEs) through different instructions. Parallel computing is done in terms of threads, the task of every thread is independent of each other. Each thread executes the same instruction to process different data. The new threads are switched in terms of warp and launched/dispatched to PEs during the memory access latency[3-4].

2 Pomelo processor

The self-developed Pomelo processor can be structurally summarized into three modules: the vector processing unit (VPU), the memory management unit (MMU), and the PE. Fig. 2 is the overview of the Pomelo processor micro-architecture. The VPU is responsible for data interaction with the external MMU, and the VPU schedules the multiple warps to fetch and decode instructions. The VPU optimizes in real-time the computing work efficiency according to the program's status. The instruction fetch unit (IFU) is responsible for reading the instructions of the 8 warps, and the switch will be triggered if memory access instruction is fetched, and a warp switching request to the warp schedule unit is sent to pause the instruction fetch pipeline. The new warp index will be arbitrated in the warp scheduler and the paused instruction fetch pipeline will be activated according to the decoder's switching request and register status.

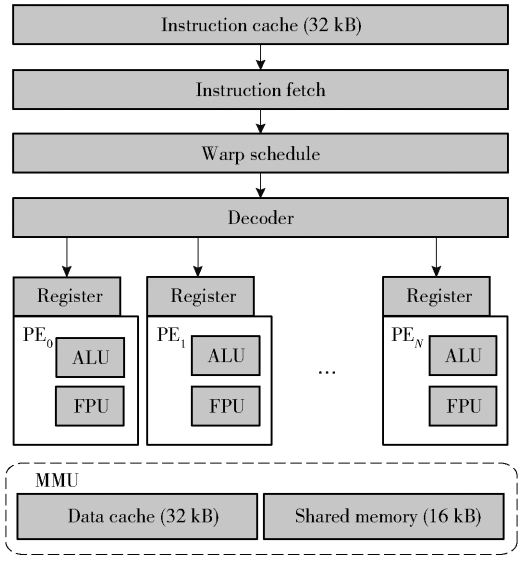

Fig. 2 Pomelo processor micro-architecture

The MMU memory unit is mainly responsible for program storage, shared memory, and data cache. The shared memory is for sharing data memory, and its main function is to save and load the data shared by multiple threads.

The PE mainly realizes the parallel operation of single instruction and multi-threaded operation under SIMT architecture, for the single instruction sent by a decoder, with the parallel execution of plural PEs. Up to at most 64 operation units are designed in the PE part, and each PE executes one thread and switches between the corresponding number of threads through different warp scheduling. Within each PE, different branching operations between threads, including program jumps and subroutine execution, are realized through the design of the respective mask units. Each of the PE operation unit has a arithmetic logical unit (ALU) and a floating point unit (FPU). The operations realized by PE operation units are list in Table 1. All floating data follows the IEEE-754 standards. The overall part of the PE coordinates the relationship between at most 64 PEs to ensure the correctness of program execution, the handling of pipeline pause, access to storage, arithmetic exceptions, and jump and call in conditional statements.

Table 1 Instruction set

Type	Instrcution	Number
Fixed-point instruction	ADD, SUB, MUL, DIV, SGE	10
Floating-point instruction	ADD, SUB, MUL, DIV, SGE	10
Logic shift	AND, OR, NOT, XOR, SLL, SRL, SRA	7
Data shift	MOV, SWAP, SETB, NEG, LDL, STL	6
Control instruction	CALL, RETN, JUMP, SETM, REVM, ENDM	6
Minimalist instruction	BADD, BGT, BMUL	3
Synchronization	SYNC	1
Other command	NOOP, HALT	2

As for the instruction set, this Pomelo processor adopts an instruction format with 24 bit length, as shown in Fig. 3. In Fig. 3, opcode is the operand corresponding to different instructions, R_d is the target register, R_a and R_b are the source register, and imm is the immediate number. The types of instructions include fixed-point instructions, floating-point instructions, logical shifts, data shifts, etc. as shown in Table 1.

23:18	17:12	11:6	5:0
Opcode	R_d	R_a	R_b/imm

Fig. 3 Instruction format

3 CNN adaptation and optimization

3.1 CNN

CNN is a neural network model widely used in image processing, pattern recognition, computer vision, and deep learning. It mainly mimics the way the human brain processes visual information.

The LeNet-5 is a classical CNN model with 1 input layer, 2 convolutional layers, 2 pooling layers, and 2 fully connected layers with a final output layer. It is used for the recognition of handwritten numbers. In the convolutional layer, different convolutional kernels could be used to process different image regions in parallel, which will speed up the process of feature extraction. The pooling layer could be computed in parallel to reduce the size of the feature map as well. These properties allow the LeNet-5 network to perform multiple computational operations in parallel in SIMT processors, taking full advantage of its parallel computing capabilities[5-7].

Based on the specific architecture and instruction set of SIMT, improvements in the operational efficiency of CNNs and the maximization of the performance potential of SIMT processors are explored, a series of adaptations for SIMT processors are carried out in terms of CNN structure parameter settings and data storage.

3.2 Optimization of CNN structure and parameters

The CNN for handwritten digit recognition is optimized and adapted to the Pomelo processor in SIMT mode, the network structure and parameters are based on LeNet-5. The convolutional operations, pooling operations, and rectified linear unit(ReLU) operations in the CNN are organized as parallel computational tasks, and these work are jointly performed by each warp in the Pomelo processor.

As shown in Fig. 4, there are $N \times N$ convolutional computation tasks, in task-level optimization strategy, each warp includes N_t convolutional computing tasks when the whole task is assigned to N_t warps, and the tasks are executed in parallel in the form of switching among warps.

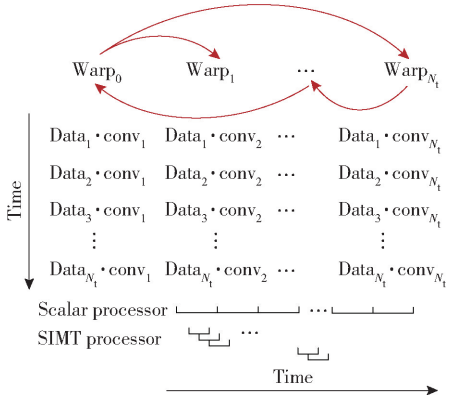

Fig. 4 Task-level parallelism

All warps are running on the 4 same PEs with a total of 128 registers. These 4 PEs perform data access and computation in parallel. The whole convolutional computing task is organized into many parallel threads, and the operation of each thread is adapted and

optimized for 4 PEs, data flow at the hardware-level is executed as shown in Fig. 5.

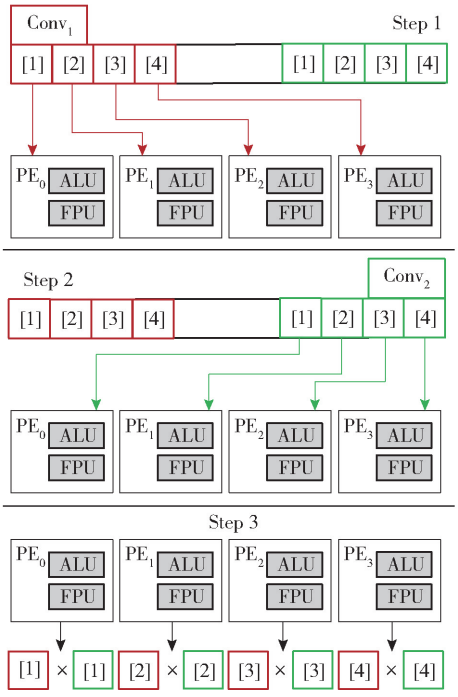

Fig. 5 Hardware-level parallelism

Step 1 An instruction (LDL R_1 R_0#0) takes out 4 data in convolution $conv_1$ and writes them into the register R_1 of 4 PEs. R_0 is the address of index data [1] of $conv_1$.

Step 2 The instruction (LDL R_2 R_{15}#0) takes out 4 data in convolution $conv_2$ and writes them into the register R_2 of 4 PEs . R_{15} is the address of index data [1] of $conv_2$.

Step 3 Calculate the dot product of convolution $conv_1$ and convolution $conv_2$ and write the result into register R_3 of the 4 PEs. Finally, the values in the 4 PEs are summed and output by the parsimonious instruction, which is the result of a complete convolution operation. The Pomelo processor reads and executes the data in parallel according to the number of PEs, which is hardware-level parallelism.

The hardware resources are 4 PEs, and 4 data can be retrieved at the same time by one data-reading operation. In order to simplify data reading and ensure that one convolution operation can be carried out normally, the PE resources should be greater than or equal to the number of convolution kernel parameters. However, multiple PE resources will inevitably lead to a waste of performance, so the convolution kernel of this network adopts the size of 2×2, which is adapted to the existing 4 PEs hardware resources of Pomelo processor.

When making stride settings, since the network takes a convolution kernel of size 2×2. If stride is taken as 1, it will lead to a data multiplexing situation, which increases the complexity of the address offset calculation in the data-accessing process. Therefore, stride is set to 2 to complete one convolution in one operation and maximize the parallel computing advantage of the Pomelo processor.

The training part of the net is done by the software side in PyTorch environment, the training set is MINIST, and the network definition is shown in Table 2. In the final training, 24 parameters of the first convolution layer, 144 parameters of the second convolution layer, 2 016 parameters of the first fully connected layer, and 840 parameters of the second fully connected layer are obtained.

Table 2 CNN network structure parameter

Net level	Net defination in PyTorch
The first convolution layer	Conv2d[1,6,kernel_size=(2,2), stride=(2,2)]
ReLU	$F(x)=\max(0,x)$
The first maxpool layer	Maxpool2d[kernel_size=2,stride=2, padding=0,dilation=1]
The second convolution layer	Conv2d[6,6,kernel_size=(2,2),stride=(2,2)]
ReLU	$F(x)=\max(0,x)$
The second maxpool layer	Maxpool2d[kernel_size=2,stride=2, padding=0,dilation=1]
The first fully connected layer	Linear[in_features=24, out_features=84]
ReLU	$F(x)=\max(0,x)$
The second fully connected layer	Linear[in_features=84, out_features=10]

In summary, the size of the native input matrix (In) is 32×32, the size of the convolution kernel of the convolutional layer is 2×2, and the stride size of both convolution and subsampling is 2. As shown in Fig. 6, M represents the number of groups of convolution kernels, and N represents the number of convolution kernels per group.

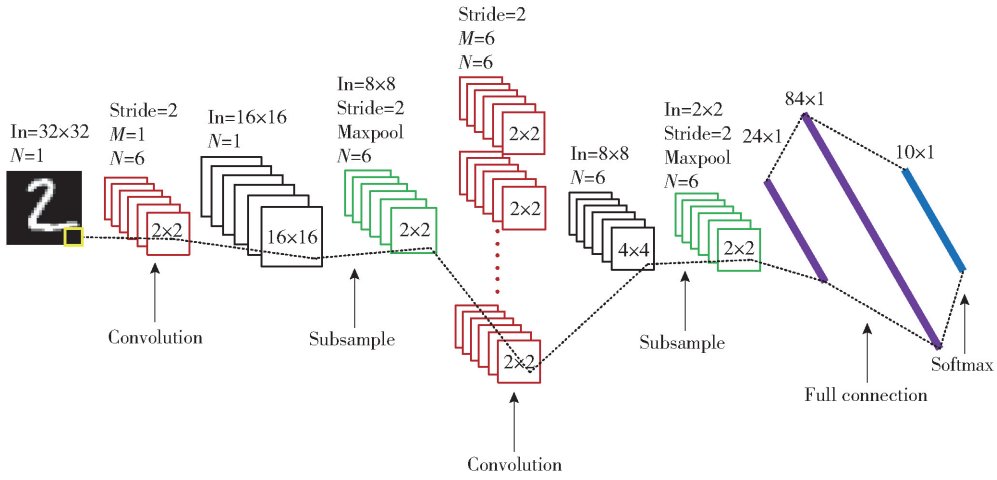

Fig. 6 CNN definition

3.3 Memory optimization

After the training of the CNN model on the software side, a total of 3 024 new weight parameters and convolution kernels are generated, and a total of 4 048 data from 1 024 data including the native matrix need to be pre-stored into the double data rate random access memory (DDR RAM), and the storage location is shown in Fig. 7. In Fig. 7, the input image data are stored in the red space. The first layer of convolutional kernel parameters are stored in the yellow space, the second layer of convolutional kernel parameters are stored in the blue space, the first layer of fully connected layer parameters are stored in the green, and the second layer of fully connected layer parameters are stored in the purple space.

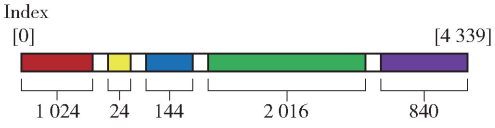

Fig. 7 Memory resource assignment

All the weight parameters are stored in Z-shaped, and the value in the R_0 register is the base address with values 0, 4, 8, and 12. PEs will retrieve the data with index [0], [1], [2], and [3] from DDR RAM accordingly. If the native input matrix is still stored sequentially, after the first layer of convolution operation, the completion of dimensionality reduction of the input matrix will result in the order of the data, as shown in Fig. 8.

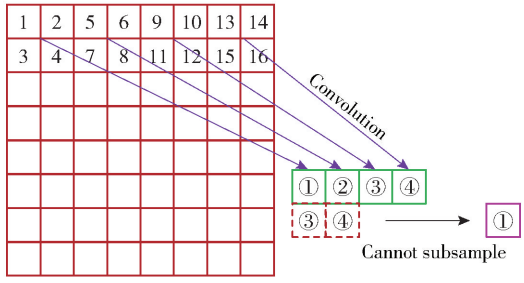

Fig. 8 Original storage structure

When subsampling and subsequent convolution operations are performed based on the data structure in Fig. 7, the data storage order is different from that of the initial native matrix, and unless the base address R_0 is modified, it is impossible to take out the two-dimensional (2D) data structure, but rather, the four neighboring indexed data are taken out by a set of row vectors, which makes it impossible to carry out subsequent subsampling and the second layer of convolution operations. For this reason, a storage structure is designed for CNN native matrices for the architectural characteristics of the Pomelo processors.

The sequential storage in Fig. 7 means that after the first layer of convolution, each pooling operation takes an additional LDL instruction to fetch the correct number. Since the whole CNN process is a chain reaction, each subsequent convolutional layer and pooling layer will require additional data-reading instructions. Considering the structural parameters of CNN, a total of 2 040 convolutional and pooling operations are required for the whole network, which

means that an additional 2 040 LDL operations are required.

The warp switching of the Pomelo processor is sensitive to LDL/STL instructions. Each LDL/STL instruction triggers the warp switching mechanism once. According to many experimental data statistics, the average time required for a single LDL/STL (including warp switching) is about 118 ns. 2 040 additional LDL operations will result in an additional 240 720 ns. Meanwhile, the additional LDL operations will necessarily include additional address offset calculations, and it is assumed that each address offset calculation can be completed with only one ADD instruction, and the execution time of the ADD instruction is 20 ns. Therefore, at least an additional 40 800 ns is required to complete the address offset calculation.

Unlike the sequential storage in Fig. 7, the novel storage structure adopts a Z-shaped storage structure. As shown in Fig. 9, every 2×2 matrix data in the native matrix is a minimum unit, and the sequential storage forms a 4×4 matrix data. Then every 4×4 matrix data as a unit, sequential storage to form an 8×8 matrix data, and so on. Benefit from the Z-shaped storage design, the processor in the process of convolution and subsampling, the new data matrix can still be obtained by the 2D structure of the number of operations, without changing the base address, to support the subsequent convolution and subsampling operations, until the full connection layer. Combined with the above, if the new storage method is adopted, the execution time can be reduced by 281 520 ns.

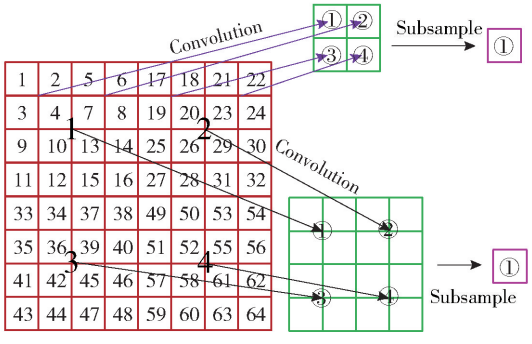

Fig. 9 Z-shaped storage structure

The full native matrix storage structure is shown in Fig. 10.

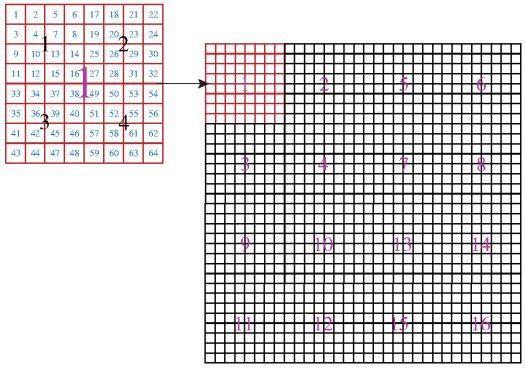

Fig. 10 Initial data and parameters of the CNN

The address index in Z-shaped storage can be solved by

$$I_{\text{addr}} = (r-1)_{\text{binary}} \otimes (c-1)_{\text{binary}} + 1 \quad (1)$$

where $\otimes$ denotes the operation of crossing and combining different bits of two numbers[8-9], r represents the row coordinates of an index, and c represents the column coordinates of an index. The computation process is shown in Fig. 11, taking $r = 5$, $c = 6$, $I_{\text{addr}} = 50$ as an example.

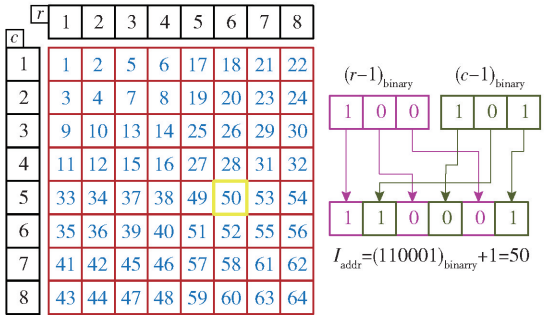

Fig. 11 Memory location

The network structure parameters and storage structure are fixed, in this paper, the hardware resource configuration of 8 warps and 4 PEs was adopted to perform the CNN recognition task, which realizes the parallel operation between 32 threads. After the modeling of DDR RAM is completed with the help of software tools, the algorithm model is developed with assembly instructions. The instructions used in the assembly process are categorized into seven types of data shifting, fixed-point instructions, floating-point instructions, other instructions, compare instructions, control instructions, and minimalist instructions, as shown in Table 3.

Table 3 Instructions used for CNN

Instruction	Type	Instruction definition
LDL	Data shifting	If (CX) R[D] = M[R[A] + im]
STL		If (CX) M[R[A] + im] = R[D]
SETB		R[A].B[7:0] = im[7:0]
ADD	Fixed-point instruction	R[D] = R[A] + R[B]
SUB		R[D] = R[A] - R[B]
MUL		R[D] = R[A] × R[B]
FADD	Floating-point instruction	R[D] = R[A] + R[B]
FSUB		R[D] = R[A] - R[B]
FMUL		R[D] = R[A] × R[B]
NOOP	Other instruction	Manual clock delay
HALT		Mark warps end
SGE	Compare instruction	R[D] = (R[A] >= R[B])
SETM	Control instruction	If (A! =0) set mask bit
REVM		Reverse current mask
ENDM		End current mask
BADD	Minimalist instruction	Data addition
BGT		Take the maximum value

After finishing the assembly program development, the number of each type of instructions is shown in Fig. 12.

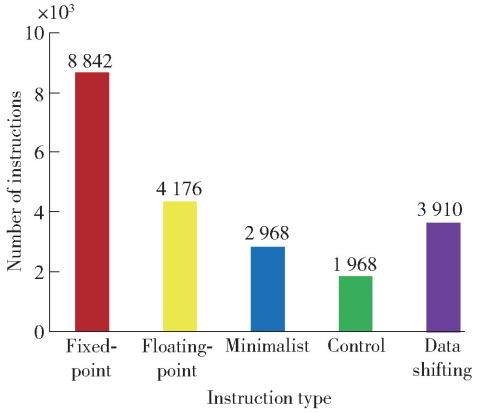

Fig. 12 Number of different instructions

4 Experiment

The input to the test case is a handwritten number 2, for details, the native input matrix is a tensor data matrix converted from a handwritten picture with the number 2. Ten simulation result weights of the softmax classifier are finally output in the waveform as 32h'3F7090D0, 32h'4084EC4C, 32h'41296EEF, 32h'40698608, 32h'C03B17C6, 32h'3CF2A800, 32h'3F137630, 32h'C0FF09D4, 32h'BED7C168, 32h'C08E0E6E in order. Converting these hexadecimal floating-point numbers to decimal gives 0.939 709 66, 4.153 844 83, 10.589 583 40, 3.648 805 62, -2.923 326 02, 0.029 621 12, 0.576 022 15, -7.973 855 97, -0.421 397 45, -4.439 261 44 in order. Table 4 shows the comparison of the output results of the software side.

Table 4 Comparison of identification weight data

Software	Pomelo processor	Error/%
0.939 707 98	0.939 709 66	0.000 001 68
4.153 842 52	4.153 844 83	0.000 002 31
10.589 583 53	10.589 583 40	0.000 000 13
3.648 805 24	3.648 805 62	0.000 000 38
-2.923 326 36	-2.923 326 02	0.000 000 35
0.029 621 62	0.029 621 12	0.000 000 50
0.576 021 62	0.576 022 15	0.000 000 53
-7.969 947 99	-7.973 855 97	0.003 907 98
-0.421 397 11	-0.421 397 45	0.000 000 34
-4.439 261 80	-4.439 261 44	0.000 000 36

The weight of the number "2" in the software model is 10.589 583 53, which is the maximum value in the result of the recognition weights, and the recognition is correct, and the inference result is that the number in the picture is "2". The weight result of the Pomelo processor is 10.58, which is basically consistent with the software model, and the average error of the recognition weights is 0.039 139%, which is analyzed and obtained from the computational error during the execution of the FPU in the hardware PE unit. A portion of the other numbers in the training set are also screened for simulation, and the error profile is obtained as shown in Fig. 13.

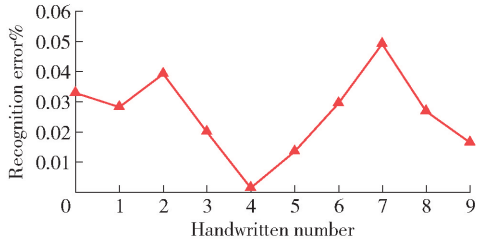

Fig. 13 Recognition weight error curve

The overall simulation average error rate is only 0.02%. The Pomelo processor recognizes the results correctly.

In terms of performance, in this test case under 4 PEs granularity of parallel computation, the simulation waveform data is counted, and the total number of running clocks of Pomelo processor is tested after excluding the cold start phase, and the acceleration ratio with single PE resources is shown in Table 5.

Table 5 Acceleration ratio

Number of PEs	Number of clocks	Acceleration rate
1	423 848	6.52
4	65 035	

Analyzing the measured acceleration ratio in Table 4 with the number and the ratio of instructions, although the increase in PE resources improved the performance of the processor, it is still impossible to ignore the impact of the overly large number of fixed-point and data shifting instructions on the acceleration ratio. Fixed-point instructions are executed by calculating the address offsets of different data during assembly language execution, while data shifting instructions involve a large number of data accesses. Therefore, the next step maybe to optimize the CNN algorithm itself by combining the hardware architectural features of the Pomelo processor as well as to improve the access conflict problem in the access path, so as to further explore the performance potential of the Pomelo processor.

5 Conclusions

In this paper, a Pomelo processor dedicated to highly parallel computing tasks is proposed, and the network parameters and data storage structure of CNN are adapted to fully utilize the advantages of SIMT computing mode.

In SIMT mode, the processor plays the highest efficiency when the convolution kernel size of CNN is equal to the number of PE. In terms of memory optimization, the Z-type storage structure is used to improve the performance of the Pomelo processor by reducing the times of memory access. The experimental result indicates that the processor that works with the Z-shaped storage structure on SIMT mode achieves a 6.52 times performance augmentation compared with scalar mode.

The future work will be focused on developing and adapting more test cases including large scale CNN and large scale matrix multiplication on the Pomelo processor.

Acknowledgements

This work was supported by the Scientific Research Program Funded by Shaanxi Provincial Education Department (20JY058).

References

[1] SANDERS J, KANDROT E. CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley Professional, 2010.

[2] NVIDIA Corporation. NVIDIA's next generation CUDA compute architecture: Fermi. Whitepaper. 2009.

[3] RIDLEY G, FORGET B. A simple method for rejection sampling efficiency improvement on SIMT architectures. Statistics and Computing, 2021, 31(3): Article 30.

[4] GAO L, XU Y L, WANG R, et al. Thread-level locking for SIMT architectures. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(5): 1121 – 1136.

[5] SUFI F. A global cyber-threat intelligence system with artificial intelligence and convolutional neural network. Decision Analytics Journal, 2023, 9: Article 100364.

[6] LI J J, WANG K, ZHENG H, et al. GShuttle: optimizing memory access efficiency for graph convolutional neural network accelerators. Journal of Computer Science and Technology, 2023, 38(1): 115 – 127.

[7] LI X M, HUANG H M, CHEN T S, et al. A hardware-efficient computing engine for FPGA-based deep convolutional neural network accelerator. Microelectronics Journal, 2022, 128: Article 105547.

[8] SHANABLEH T. Recursive quad-tree block partitioning for data embedding in images. SN Computer Science, 2020, 1(6): Article 309.

[9] ROSELINKIRUBA R, SHARMILA T S. A novel data hiding by image interpolation using edge quad-tree block complexity. The Visual Computer, 2021, 39(1): 59 – 72.

(Editor: Wang Xuying)