doi:10.3772/j.issn.1006-6748.2023.01.006

IOPS: computational graph optimization based on inter-operators parallel scheduling①

XIE Xiaoyan(谢晓燕)*, XU Hao②*, ZHU Yun**, HE Wanqi*
(*School of Computer, Xi'an University of Posts and Telecommunications, Xi'an 710121, P. R. China)
(**School of Electronic Engineering, Xi'an University of Posts and Telecommunications, Xi'an 710121, P. R. China)

Abstract

To improve the inference efficiency of convolutional neural networks (CNN), the existing neural networks mainly adopt heuristic and dynamic programming algorithms to realize parallel scheduling among operators. Heuristic scheduling algorithms can generate local optima easily, while the dynamic programming algorithm has a long convergence time for complex structural models. This paper mainly studies the parallel scheduling between operators and proposes an inter-operator parallelism schedule (IOPS) scheduling algorithm that guarantees the minimum similar execution delay. Firstly, a graph partitioning algorithm based on the largest block is designed to split the neural network model into multiple subgraphs. Then, the operators that meet the conditions is replaced according to the defined operator replacement rules. Finally, the optimal scheduling method based on backtracking is used to schedule the computational graph. Network models such as Inception-v3, ResNet-50, and RandWire are selected for testing. The experimental results show that the algorithm designed in this paper can achieve a 1.6× speedup compared with the existing sequential execution methods.

Key words: compile optimization, convolutional neural network(CNN), inter-operator parallelism schedule(IOPS), operator replacement

0 Introduction

In recent years, with the widespread application of neural networks in computer vision, speech recognition, and games, the computational complexity of their models has increased accordingly. Currently, industry and academia are concerned with the efficient use of hardware resources. In order to make full use of hardware resources and reduce the training and inference time of the model, existing methods have been explored in many aspects. Ref. [1] used model pruning to reduce the number of effective weights and the accuracy of the parameters. Ref. [2] proposed parameter quantization to reduce the number of bits per weight, thus accelerating inference efficiency. Ref. [3] adopted data and model parallelization[4] to train and infer network models. Multi-operator parallelism has achieved excellent optimization results as it can comprehensively consider various optimization factors.

At present, there are two research directions of multi-operator parallelism: intra-operator parallelism and inter-operator parallelism. Intra-operator parallelism mainly uses the underlying acceleration library cuDNN[5] provided by hardware suppliers to find the optimal parameters of a single operator in the network structure under the current configuration. However, intra-operator parallelism can not provide maximum parallelism, which results in suboptimal model runtime performance. Secondly, the serial execution employed by the underlying acceleration units of the deep learning framework[6] misses the opportunity to further improve performance. A series of optimization algorithms[7] can be used between operators to effectively combine intra-operator and inter-operator parallelism. Therefore, many scholars have conducted much research on the inter-operator of computational graphs.

The existing inter-operator parallel methods mainly use heuristic algorithms to realize parallel scheduling of multiple neural network operators. Ref. [8] proposed an improved heuristic graph partitioning algorithm machine learning-Metis (ML), which considers

① Supported by the National Key Research and Development Project of China (No. 2020AAA0104603), and the National Natural Science Foundation of China (No. 61834005, 61772417), and the Shaanxi Province Key R&D Plan (No. 2021GY-029).
② To whom correspondence should be addressed. E-mail: qaz1352plm@163.com.
Received on Sep. 5, 2022

the memory multiplexing and intermediate data from data flow graphs during partitioning. It provides an efficient parallel strategy for deep neural networks (DNN). Ref. [9] used graph substitution to optimize the computation graph. Compared with the original computation graph, the optimized computation graph will optimize the scheduling process in parallel and uses a sampling heuristic algorithm to speed up the search process. However, the greedy parallelism of the heuristic algorithm will lead to a suboptimal scheduling scheme. The parallelism of algorithm optimization will have a good effect on the network of medium-structured networks, but the effects are not obvious for small networks.

Ref. [10] introduced a strategy to parallelize DNNs in the Sample, Operator, Attribute, and Parameter dimensions, which is a more comprehensive DNN parallelization strategy search space. It uses a depth-first search to explore the space, but its search model optimal strategy takes 0.8 and 18 h respectively. Furthermore, machine learning practitioners[11] rely on their domain knowledge and use manual cues (e.g., explicit device placement strategies) to guide compiler decisions. Reinforcement learning-based (RL) methods[12] outperformed human experts and heuristics. However, they require many computing resources and training time to find the optimal schedule.

To sum up, the methods mentioned above use different heuristic algorithms to seek the optimal solution, but the solution result is not necessarily the optimal solution in the global scope. Using methods based on machine learning and dynamic programming to expand the search space, approximate global optimal solutions can be obtained. However, its algorithm does not make full use of general sub-scheduling when searching the entire model architecture, and it is difficult to use historical optimization results. When the model scale is enormous, optimizing the entire computational graph at one time will cause the problem of long algorithm convergence time.

The rest of the paper is organized as follows. Section 1 discusses related work, and Section 2 describes the optimization process of this paper and the scheduling problem for neural networks and details the two algorithms and the parallel replacement strategy in this paper. Section 3 describes the specific details of the experiments in terms of the neural network, hardware configuration, etc., provides the experimental results and related analysis in this paper, and finally summarizes the entire experiment and puts forward prospects in Section 4.

1 Related work

Multi-operator parallelization is a leading research direction to achieve efficient reasoning. Operators at different stages are scheduled in different threads. Existing research on implementing operator parallelism mainly focuses on intra-operator parallelism and inter-operator parallelism.

Existing deep learning frameworks usually utilize intra-operator parallelism to combine the arithmetic operations (matrix multiplication) of a single neural network operator with serial execution adopted by the underlying acceleration unit, missing opportunities to further improve performance. Ref. [13] used the cost model-based neural network search method, which can automatically find the high-performance tensor parameters corresponding to the current operator for different hardware devices to guide the reasoning execution of the model. However, with the continuous improvement of hardware computing power, only intra-operator parallelism cannot provide sufficient parallel scheduling optimization solutions for hardware devices, so a series of optimization algorithms can be used.

The existing methods of inter-operator parallelism mainly use heuristic and neural network algorithms to schedule the parallel execution of multiple operators on the underlying hardware at the same time. Ref. [14] proposed a greedy strategy to execute all available DNN operators directly on hardware devices to maximize resource utilization. Ref. [15] proposed a nimble deep learning (DL) execution engine, which can run graphics processing unit (GPU) tasks in parallel with minimal scheduling overhead. Ref. [16] only used a finite convolution operator fusion strategy of the same type of operator (Conv) for fusion. It does not take into account substitutions between operators of different types. Ref. [17] explored the scheduling space through a novel cross-operator dynamic programming algorithm. It finds highly optimized schedules with low search costs, which further optimizes schedule inference latency between operators. However, the algorithm is a one-time exploration and optimization of the entire computational graph of the neural network model, which results in a geometrically exponential increase in scheduling time when scheduling some large models.

Therefore, in this paper, under the condition of equal model inference scheduling delay, a globally optimal inter-operator parallelism schedule (IOPS) algorithm with fast convergence speed is designed.

2 Methods

2.1 Optimization process and schedule description

This paper proposes an inter-operator parallelism schedule (IOPS) to address the above problems. The optimization process of this schedule is shown in Fig. 1. First, taking the initial computational graph as input, a computation graph splitting algorithm based on the largest block is designed, which recursively divides the original complex computational graph into smaller subgraphs. Second, according to the operator characteristics of the neural network computation graph, the operator replacement and operator parallel strategies are proposed. Third, in the scheduling process, select the optimal scheduling method (parallel or replacement) as the intermediate results are saved. Finally, each ProTimes of the scheduling algorithm is added to get the scheduling latency and strategy of the whole set of models.

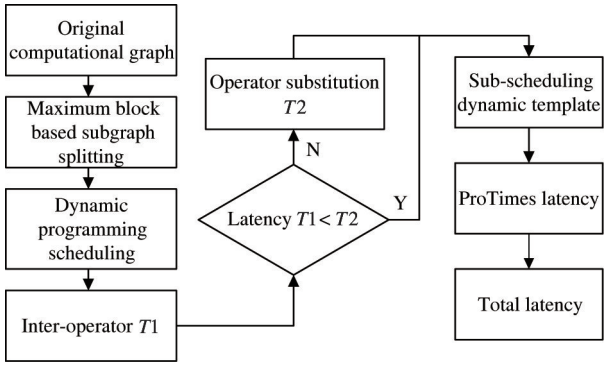


Fig. 1 Overall optimization process

As neural networks evolve, computation graphs are used to represent the computations required for a compiled deep learning model. The computational graph is a directed acyclic graph, which is denoted as $G(V,E)$, where V represents the operator in the computation graph, and $E(a,b)$ is used to connect the operators in the computation graph, representing the operator dependency between a and b.

In this paper, a dynamic programming scheduling based on dynamic sub-scheduling is used. To better describe this problem, the operator dynamic partitioning problem is formally defined in the following form. S, $S1$ and $S2$ represent the set of operators. $S = \{b, c, d, e, f, g\}$, $S1 = \{c, e, f, g\}$, $S2 = \{f, g\}$.

As shown in Fig. 2, S represents a set of operators in the computational graph. $S1$ is the intermediate set of S. Subsequent nodes to the set of $S1$ should not be in S, and for all scheduling, a set of scheduling nodes must exist in a subset of $S1$. Enumerate all non-empty subsets of $S1$, if the $S2$ set is the optimal scheduling set, it needs to find the optimal scheduling $S\text{-}S2 = \{b, c, d, e\}$. Repeating the above steps in this way can reduce the computational complexity, and finally obtain the entire computational graph.

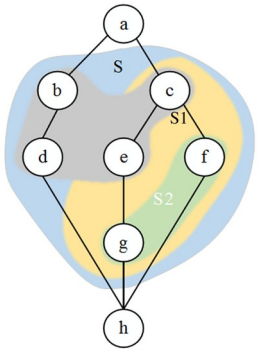


Fig. 2 Description of the parallel scheduling process inter-operator

For the scheduling delay evaluation of computational graphs, this paper uses an improved dynamic programming algorithm to schedule subgraphs. Eq. (1) is used as the state transition equation for dynamic programming scheduling, where G' represents the subgraph generated by the graph partitioning algorithm, and the scheduling delay $ProTimes(L')$ is obtained through IOPS sub scheduling. Therefore, the problem turns into finding the scheduling delay that minimizes the subproblem $Cost(G' - L')$.

$$CostModel(G') = \min(ProTimes(L') + Cost(G' - L')) \quad (1)$$

$$CostModel(G) = \sum_{i=1}^{n} (ProTimes[S_i]) \quad (2)$$

The total delay of the entire computational graph scheduling can be obtained by adding the delay time of each Pro, as shown in Eq. (2). The measured delay is used as a cost model to guide the optimization of a given process $Pro(S_i)$ and the corresponding alternative parallel strategies. The optimal scheduling of the entire computational graph can be obtained. Assume that the total delay of the final computation graph scheduling is the sum of the delays of each *Pro*. That is, the scheduling delay for the next *Pro* is not affected by the previous *Pro*. Scheduling between *Pros* is independent of each other. Eq. (2) will be experimentally verified in subsection 3.7.

2.2 Model splitting algorithm

Many state-of-the-art neural network models are too large to optimize the entire computational graph through search. This paper uses a block-based model splitting algorithm to partition the computational graph

into smaller disjoint subgraphs recursively. The maximum block is defined as the maximum number of parallel operators that a thread can operate in parallel. In this experiment, the maximum number of operators in each block is 50. The split subgraph can be scheduled using a parallel optimization strategy.

Because operator replacement cannot be performed on any two sub-blocks, the goal of splitting is to minimize the number of operator replacements that span the two subgraphs. For each operator $S_i \in G$ define $cap(S_i)$ as mapping to number of operator substitutions for at least one input edge and one output edge of operator S_i. Disable operator substitution if the graph is split using operator split. The model splitting problem is mapped to a minimum point cut problem by using $cap(S_i)$ as the weight of each operator. Algorithm 1 shows a model splitting algorithm using the most extensive block algorithm as an example. After running the dynamic template scheduling algorithm to optimize individual subgraphs, IOPS re-stitches the optimized subgraphs together to form a complete computational graph. Finally, a local backtracking search is performed around each split point to obtain operator substitutions across the split.

Algorithm 1 A blocks-based graph split algorithm

Input: Initialized computational graph
Output: Calculation graph after splitting
function *ModelSplit*(G)
  if $|G| \leqslant$ max_part_size **then**
    return G
  else
  //MIN-SPLLT(·) #Returns the operator replacement
    P = MIN-SPLIT(G)
    $G_1 = \{S_i \in G \mid S_i$ can be accessed from $P\}$
    $G_2 = G - G_1$
    return {*ModelSplit* ($G1$), *ModelSplit* ($G2$)}
  end if
end function

Although the most block-based model splitting algorithm is effective and achieves good performance for all models used in the experiments, it is not a universal model splitting algorithm for all models.

2.3 Operator replacement

In the operator replacement strategy, each replacement consists of a source operator and a target operator. The source operator can be mapped to a sub-operator in the model computational graph. The target operator defines how to create a new sub-operator to replace it.

The source operator defines the structure of the replacement valid operator. The target operator describes how to construct a new operator to replace the mapped operator. For each newly created operator, the target operator defines how to set parameters and calculate weights using the parameters and weights from the source operator. The outer edges of the destination operator should correspond to every outer edge of the source operator, and any outer operators initially connected to the map operator in the source operator should now be connected to the corresponding operator in the destination operator.

Each node in a source operator is associated with a type and can only be assigned to operators of the same type. Source operators can also include generic nodes, which are helpful when the operator type does not affect the replacement process and when the source operator describes multiple similar replacement scenarios. In addition to type constraints, source operators can also contain other convolution size constraints on one or more operators to restrict replacement further.

Fig. 3 shows the method of operator replacement. The first step is to replace Conv2 and Conv3 operators with Conv4 operators. The next step is to replace the Conv6 and add operators with the Conv7 operator. After the Conv4 operator, the replaced operator will be split to maintain the dependencies between operators.

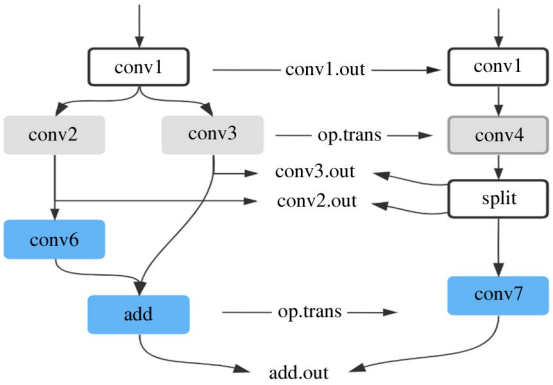


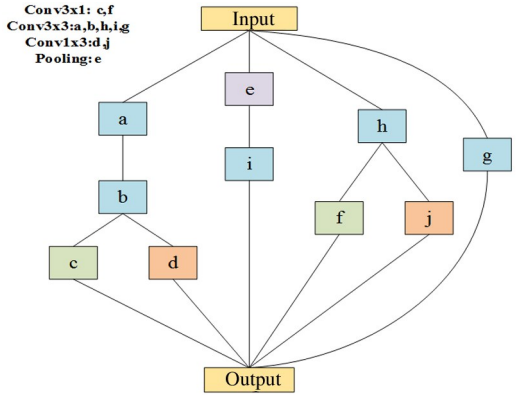
Fig. 3 The method of operator replacement

2.4 Parallel scheduling

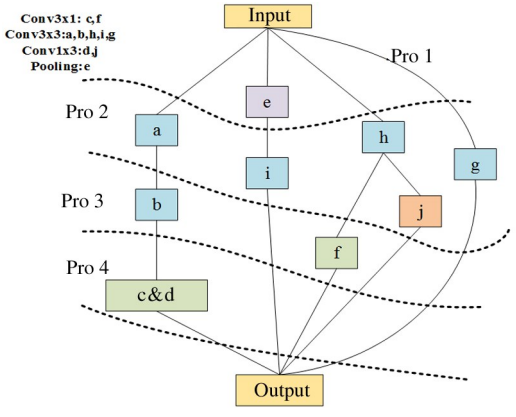
The parallel scheduling between operators based on dynamic templates will automatically find an optimal schedule for the current computational graph under the given cost model, underlying hardware, and inference parameters. Compared with the traditional dynamic programming scheduling, this paper uses the backtracking technology of the dynamic template, which greatly reduces the convergence time of finding the optimal scheduling.

Fig. 4 shows the changes in the execution order of model scheduling before and after using operator re-

placement and the IOPS scheduling algorithm. Take an Inception module in the Inception-v3 model[18] as an example. After using the model splitting algorithm based on the largest block, the module to be scheduled is obtained by taking the Inception module as a subgraph. As shown in Fig. 4(a), the nodes (a, b, h, i, and g) indicate operators with a convolution size of 3 × 3, and the nodes (c and f) represent a convolution operator with convolution size of 3 × 1, the nodes (d and g) represent convolution operators with a convolution size of 1 × 3, and the nodes are model grouping operators. Due to the different processing methods of different back-end IOPS algorithms, this paper selects the GPU NVIDIA GeForce RTX 2080ti, the IOPS scheduling algorithm adopts the sub-Pro scheduling method. As shown in Fig. 4(b), the scheduling of nodes (e) is divided into Pro1, which means that during this process, nodes (e) occupy a single GPU thread for scheduling. In Pro4, expand the convolution operator of the node(d) to 3 × 3 size, replace the original nodes (c) and (d) nodes, get the replaced operator [c & d], compare the parallelism and replacement latency, and



(a) Before scheduling optimization



(b) After IOPS optimization

Fig. 4 Changes in the execution order of model scheduling before and after using operator replacement

choose the best result. This minimizes scheduling delays across the entire computation graph. The specific IOPS scheduling Algorithm 2 is as follows.

Algorithm 2 Parallel scheduling

Input: Computation graph G

Output: G Schedule

1: **initialization** Times $[S_1] = \infty$, Policy$[S_1] = \varnothing$# S_1 is a subset of the operator subset S
2: **function** IPS(S)
3: V = Computational graph of all operators
4: **For** All end operators set S_1, Policy P do
5: L_{S_1}, P_{S_1} = **Pro** (S_1)
6: S_i = **SplitModel**(S_1), $(i = 1, 2, \cdots, i)$
7: $Template[S_i] = Times(S_i)$
8: **if** $group[S_1] == list[S_i]$ then
9: **return** $Template[S_i]$#assign
10: $L_{S_{i_{par}}} = Times(S_{i_{par}})$
11: **if** S_i can trans? $L_{S_{i_{trans}}}$:$L_{S_{i_{trans}}} = \infty$
12: **if** $L_{S_{i_{trans}}} \geqslant L_{S_{i_{par}}}$? $L_{S_{i_{par}}}$: $L_{S_{i_{trans}}}$
13: $L_{S'}$ = **IOPS** $(S - S_{i-1}) + L_{(S)}$
14: **if** $L_{S'} < Times[S]$ then #Update strategy
15: $Times[S] = L_{S'}$
16: $Policy[S] = (S_{i-1}, Policy[S_{i-1}])$
17: **end if**
18: **end for**
19: **return** $Times(S)$
20: **end function**

In optimizing the scheduling search, use $Template[\,]$ to save the sub-scheduling in the current scheduling process if the scheduling set of a group of sub-graphs changes. When making policy selection, the current optimal *Policy* will be selected according to the calculated *Times* value. First, whether the current operator can perform operator replacement operations will be selected according to the strategy defined in subsection 2.3. If it is judged as no, $L_{S_{i_{trans}}}$ is assigned as infinite, which means irreplaceable. Otherwise, return the delay of the current operator replacement to $L_{S_{i_{trans}}}$, in lines 10 – 12, IOPS will return the optimal delay and policy of the current operator set according to $L_{S_{i_{trans}}}$ and $L_{S_{i_{par}}}$. Line 13 is the *CostModel* defined in Eq. (2), which performs dynamic programming to explore the scheduling delay of the remaining operators, with the aim of minimizing the scheduling delay of the scheduling sub-problem $(S - S_i)$. Lines 14 – 16 will automatically update the current optimal scheduling time and strategy according to the $L_{S'}$ obtained from the defined *CostModel* and finally return the optimal scheduling delay and strategy.

3 Experimental results and analysis

In order to verify that IOPS is a general computational graph scheduling algorithm, five types of popular neural network model architectures are used for experiments. The specific parameter configurations are shown in Table 1.

Table 1 Structure parameters of neural network model

NetWorks	Blocks	Convs	Ops	Type
Inception-v3[18]	13	41	102	Conv
ResNet-50[19]	16	20	83	Conv
NasRNN[20]	1	——	30	RNN
NasNet[21]	1	264	1128	SepConv
RandWire[22]	456	112	456	SepConv
SqueezeNet[23]	12	30	50	Conv
BERT[24]	8	——	113	Attention

Experimental environment settings: Python 3.7, CUDA 11.1, cuDNN version 8.0.5, ubuntu version 18.04, CPU 6x Xeon E5 -2678 V3, GPU NVIDIA GeForce RTX 2080ti, professional inference engine TensorRT version 7.0.0.11.

3.1 Ablation experiment

This section conducts a series of ablation experiments on each module of the algorithm for different networks. Different neural network models use three methods for scheduling. The scheduling latency results obtained by each optimization method are shown in Table 2. TS means the time delay obtained by the scheduling algorithm when only replacement is performed. PS means the scheduling delay obtained by only performing operator parallelism.

Table 2 Latency of different methods

Models	Latency/ms		
	TS	PS	TS + PS
NasRNN	——	0.71	**0.71**
NasNet	20.48	15.25	**14.23**
SqueezeNet	0.89	0.79	**0.64**
Randwire	3.91	3.64	**3.53**
Inception-v3	4.6	4.53	**4.03**
BERT	——	1.38	**1.41**
ResNet-50	2.24	1.93	**1.85**

It can be seen from the results that the delay obtained by using the PS method is generally smaller than the scheduling delay obtained by TS, because TS only replaces convolution operators that meet the constraints, and its applicable range is limited. PS can satisfy the parallel execution between different source operators (such as Conv, Add and Relu). This method (TS + PS) combines these two optimization methods. Selecting the best scheduling scheme in the process of global optimization can get the best overall scheduling delay.

3.2 End-to-end scheduling methods

This paper compared four different scheduling methods: Sequential (SQ), Greedy (GE), TensorRT (TR), and IOPS. Based on the above environment settings. The maximum number of operators in each experiment process is set to 4 and the maximum number of processes is 8, and the experimental data is repeated 10 times to take the mean value for statistics. When the batch size is 1, SQ executes each operator in turn according to the topological shape of the model computation graph. GE is putting all the operators that can be executed in the same process in parallel in the current state. As a professional inference engine, TR performs efficient inference scheduling according to its internally defined execution order. The IOPS scheduling in this paper is based on the defined operator replacement and dynamic template scheduling algorithms, and the operators in the computation graph are executed in parallel among the operators. All four scheduling methods are executed on the same execution engine. In this paper, 10 times of scheduling are used to obtain the average value, and finally the standard throughput of each model delay is shown in Fig. 5.

It can be seen from Fig. 5 that the inference delay of the IOPS scheduling algorithm is generally lower than that of the other three scheduling methods. For the network structure with the Inception module and ResNet residual block, because the operators of the model design have a high degree of parallelism and the network model belongs to a deep neural network, the communication overhead between operators can be greatly reduced through the parallel replacement strategy and achieved a more obvious acceleration. For RandWire and SqueezeNet, which are obtained by neural network architecture search, in order to make full use of hardware resources, a large number of parallel operators are usually generated, so the algorithm in this paper can achieve friendly acceleration for this model. For the bidirectional encoder representation from transformers (BERT) series of network models, due to the sequential nature of its structure and fewer branches, the IOPS method only performs limited parallelism, and the actual acceleration effect is modest.

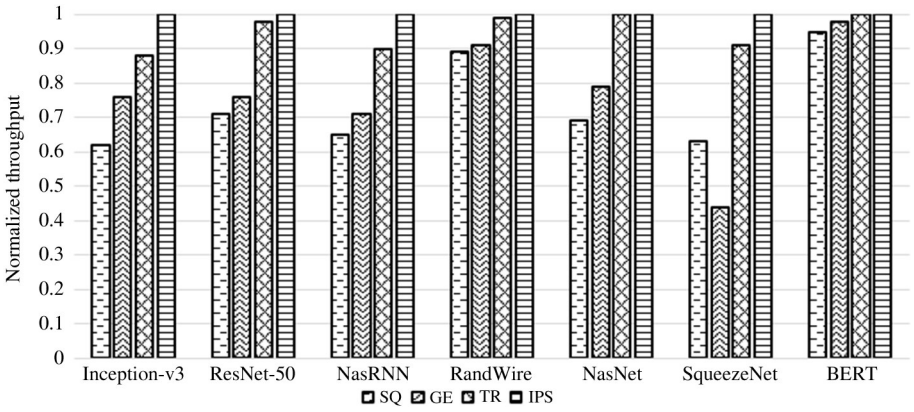


Fig. 5 Operator scheduling inference delay

3.3 Comparison of optimization time

For scheduling reasoning of neural networks, model optimization convergence delay is also an important part. Therefore, based on the test environment in this paper, the optimized time of different networks under the same underlying hardware environment is tested.

The model optimization time is shown in Table 3. Compared with the heuristic enumeration search, the experiment setup of this article sets the maximum parallel operator quantity for (*Pro*) to be 4, and the maximum number of processes to be processed is 8, thus reducing the search range. Compared with some models with simple structures in Ref. [17], the optimization time of the Inception-v3 model is increased by nearly 5 times, and the optimization time of the ResNet-50 and SqueezeNet models is increased by 2 times. For the NasNet model, an order-of-magnitude improvement has been achieved. Ref. [25] used the backtracking search algorithm based on the *Cost Model*. In this paper, $Q = 20$ and $\varphi = 1$ are used to calculate the convergence time of the model. For models with complex structures such as the RandWire model, the number of replacement strategy operators in this paper is reduced to 406. Therefore, the optimization time of the algorithm has been greatly improved. In addition, due to the time consumption of the selection strategy, the convergence time of the algorithm in this paper is slightly higher than that in Ref. [9] for the complex model BERT.

Table 3 Optimization time of different networks

Model	Ref. [17]	Heuristic	Ref. [10]	Ref. [25]	Ref. [9]	This work
NasRNN	——	——	——	>1 h	20.59 s	49 s
NasNet	10 min	>1 h	>1 h	>1 h	233.34 s	2 s
SqueezeNet	10 s	>1 h	——	——	——	5 s
RandWire	20 min	>1 h	>1 h	——	——	7.2 min
Inception-v3	10 s	12.8 min	40 s	18.75 s	4.38 s	2 s
ResNet-50	5.25 s	>1 h	60 s	8.75 s	1.14 s	2 s
BERT	——	>1 h	——	20 s	11.69 s	17 s

3.4 Different hardware backends

To verify the applicability of the algorithm in this paper on different hardware backends, this section conducts experiments on the parallel delay allocation of the Inception-v3 network on three different underlying hardware GPUs.

As shown in Table 4, TotTime represents the total delay. For the Inception-v3 model, the model splitting algorithms in this paper split it into 13 sub-blocks (labeled 0-12). For SQ scheduling, the scheduling resources of the model are mainly used for the Inception module, which is sub-block 11. For the IOPS inter-operator scheduling algorithm, multiple operators' parallel numbers and methods can be adjusted according

Table 4 Different hardware backends

	Method	TotTime	Blocks	Block	Time
Tesla K80	SQ	35.31 ms	13	[11]	4.51 ms
	IOPS	25.00 ms	13	[0]	4.34 ms
RTX 2080ti	SQ	6.47 ms	13	[11]	0.84 ms
	IOPS	4.03 ms	13	[9]	0.51 ms
RTX 3090	SQ	6.04 ms	13	[11]	0.88 ms
	IOPS	3.33 ms	13	[8]	0.44 ms

to the maximum availability of resources during model scheduling for different execution backends.

3.5 Scheduling delay of different batch sizes

The batch sizes of the network will affect the scheduling delay of the model. The average delay of 10 experiments was selected for the experiment. As shown in Fig. 6, the IOPS method is much lower than the other two scheduling methods under different batch sizes. GE exhibits better scheduling delay than SQ in most cases. When the batch size is 8, the GE delay obtained by the experiment is the largest. By analyzing the scheduling delay of each Inception block in Inception-v3, when the batch size is 8, the GE has resource contention in the early stage of Pro, so the underutilized situation can lead to a GE delay higher than SQ scheduling. The IOPS method proposed in this paper is usually lower than the other two scheduling methods.

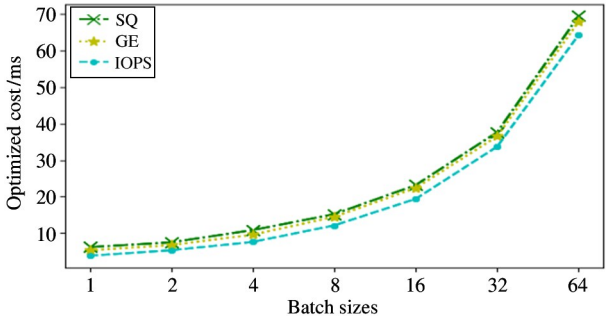


Fig. 6 Different batch sizes

The algorithm in this paper allocates different parallel branch sizes adaptively for different batch sizes, so it is also applicable for different batch sizes.

3.6 Other performance index parameters

By comparing different metrics of SQ and IPS, the IPS method reduces delay and improves device utilization. The method of utilization rate is shown in Eq. (3):

$$U_m = \frac{GFLOPs}{Latency}\ (\text{GFLOPs/ms}) \tag{3}$$

Table 5 presents several performance metrics. For Inception-v3 and ResNet-50 networks, the number of operators in the model is reduced after the operator replacement strategy is adopted, so the memory occupied and the number of kernel launches are reduced. For the memory and kernels occupied by the model, the operator replacement strategy in this paper reduces the number of operators in the model as a whole, and the final memory usage is also reduced. In the model scheduling process, the number of operators is reduced by using operator replacement operations, which means that the number of operators sent to the CUDA stream is reduced. For a fixed model, the calculations for the same model are a fixed value. Therefore, compared with SQ scheduling, the giga floating-point operations per second (GFLOPs) of each model do not vary, the utilization rate of the hardware U_m is calculated. Therefore, the algorithm in this paper has strong applicability to deep neural networks with convolution operations.

Table 5 Comparison of performance parameters between SQ and IOPS

Models	GFLOPs		Memory/MB		Kernels		Latency/ms		U_m	
	SQ	IOPS	SQ	IOPS▽	SQ	IOPS▽	SQ	IOPS▽	SQ	IOPS△
NasRNN	2.27	2.27	0.72	**0.64**	5	5	0.83	**0.71**	2.73	**3.19**
NasNet	23.12	23.12	296.8	296.8	972	**965**	20.60	**14.23**	1.12	**1.62**
SqueezeNet	1.22	1.22	14.24	**13.15**	50	**46**	1.01	**0.64**	1.21	**1.90**
RandWire	8.06	8.06	106.0	**105.9**	410	**406**	3.92	**3.53**	2.05	**2.28**
Inception-v3	5.78	5.78	20.6	**18.03**	119	**106**	6.47	**4.03**	0.89	**1.27**
ResNet-50	2.51	2.51	9.50	**8.97**	86	**82**	2.58	**1.85**	0.97	**1.36**

3.7 ResNet-50 delay per block

Assume that the delay of the entire computational graph is equal to the sum of the delay of each *Pro*. To verify the hypothesis of Eq. (2) random sampling is performed on each process of the computational graph, and then the sampled results are summed to evaluate the actual delay of the entire computational graph. The predicted delay is obtained by adding the scheduling delays of all processes. The results of the actual and predicted delays are shown in Fig. 7. By calculation, the sum of the difference between the predicted value

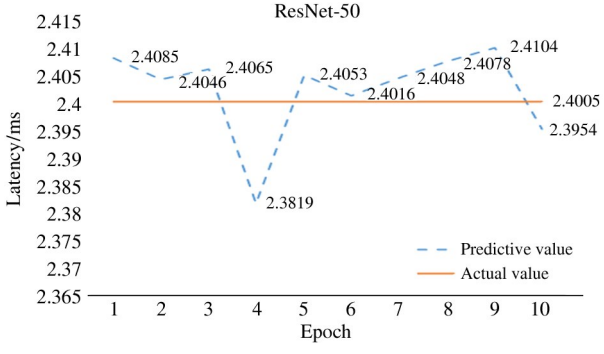


Fig. 7 The results of the actual and predicted latency

and the actual value of the 10 times of data is 0.022, and the final average precision error is 0.002.

4 Conclusion

The algorithm designed in this paper is a general neural network scheduling algorithm. Firstly, a graph partition algorithm based on the largest block is designed to divide the neural network model into multiple subgraphs. Then, operators that meet the conditions are replaced according to the defined operator replacement rules. Finally, the optimal scheduling method based on backtracking is used to schedule the computation graph. Through analysis of scheduling delay and convergence time in the experimental part, it can be concluded that the proposed algorithm can achieve 1.6 × acceleration for deep neural networks such as NasNet and SqueezeNet compared with SQ scheduling. For RandWire model, compared with SQ scheduling, only PS strategy can achieve 1.1 × acceleration. In this experiment, most models can be optimized within 10 min. Therefore, the algorithm proposed in this paper can be widely used in the research of parallel architecture of neural network models. In the future, it is considered to optimize the algorithm replacement rules and perform deeper optimization on the model calculation graph level.

References

[1] GUPTA S, AGRAWAL A, GOPALAKRISHNAN K, et al. Deep learning with limited numerical precision[C]// International Conference on Machine Learning. New York: ACM, 2015: 1737-1746.

[2] LIU Z, LI J, SHEN Z, et al. Learning efficient convolutional networks through network slimming[C]// Proceedings of the IEEE International Conference on Computer Vision. Venice: IEEE, 2017: 2736-2744.

[3] HE X, YAO Y, CHEN Z, et al. Efficient parallel A * search on multi-GPU system[J]. Future Generation Computer Systems, 2021, 123: 35-47.

[4] TAVARAGERI S, SRIDHARAN S, KAUL B. Automatic model parallelism for deep neural networks with compiler and hardware support[EB/OL]. (2019-06-11)[2022-11-05]. https://arxiv.org/pdf/1906.08168.pdf: arXiv.

[5] CHETLUR S, WOOLLEY C, VANDERMERSCH P, et al. CUDNN: efficient primitives for deep learning[EB/OL]. (2014-12-18)[2022-11-05]. http://arxiv.org/pdf/1410.0759.pdf: arXiv.

[6] ABADI M. TensorFlow: learning functions at scale[C]// Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming. Nara: ACM, 2016: 1-1.

[7] MA L, XIE Z, YANG Z, et al. Rammer: enabling holistic deep learning compiler optimizations with {rTasks} [C]// The 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). Banff: USENIX, 2020: 881-897.

[8] LI Z, CHEN L, XU N, et al. Parallel computation of deep neural network optimized based on ML-Metis[C]// 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC). Chongqing: IEEE, 2020: 1126-1129.

[9] FANG J, SHEN Y, WANG Y, et al. Optimizingdnn computation graph using graph substitutions[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2734-2746.

[10] JIA Z H, ZAHARIA M, AIKEN A. Beyond data and model parallelism for deep neural networks[C]// Proceedings of Machine Learning and Systems. Palo Alto: ACM, 2019: 1-13.

[11] POURGHASSEMI B, ZHANG C, LEE J H, et al. On the limits of parallelizing convolutional neural networks on GPUs[C]// Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures. New York: ACM, 2020: 567-569.

[12] ZHOU Y, ROY S, ABDOLRASHIDI A, et al. Transferable graph optimizers for ML compilers[J]. Advances in Neural Information Processing Systems, United States, 2020, 33: 13844-13855.

[13] CHEN T, MOREAU T, JIANG Z, et al. {TVM}: an automated {end-to-end} optimizing compiler for deep learning[C]// The 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). Carlsbad: USENIX, 2018: 578-594.

[14] TANG L, WANG Y, WILLKE T L, et al. Scheduling computation graphs of deep learning models on manycorecpus[EB/OL]. (2018-07-16)[2022-11-05]. http://arxiv.org/pdf/1807.09667.pdf: arXiv.

[15] KWON W, YU G I, JEONG E, et al. Nimble: lightweight and parallel GPU task scheduling for deep learning[J]. Advances in Neural Information Processing Systems, United States, 2020, 33: 8343-8354.

[16] JIA Z, LIN S, QI C R, et al. Exploring hidden dimensions in parallelizing convolutional neural networks[C]// ICML. Stockholm: ADS, 2018: 2279-2288.

[17] DING Y, ZHU L, JIA Z, et al. IOS: inter-operator scheduler for CNN acceleration[J]. Proceedings of Machine Learning and Systems, 2021, 3: 167-180.

[18] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the inception architecture for computer vision[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016: 2818-2826.

[19] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]// Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016: 770-778.

[20] ZOPH B, LE Q V. Neural architecture search with reinforcement learning[EB/OL]. (2017-02-15)[2022-11-05]. http://arxiv.org/pdf/1611.01578.pdf: arXiv.

[21] ZOPH B, VASUDEVAN V, SHLENS J, et al. Learning transferable architectures for scalable image recognition[C]// Proceedings of the IEEE Conference on Computer

Vision and Pattern Recognition. Salt Lake City: IEEE, 2018: 8697-8710.
[22] XIE S, KIRILLOV A, GIRSHICK R, et al. Exploring randomly wired neural networks for image recognition[C] //Proceedings of the IEEE/CVF International Conference on Computer Vision. Long Beach: IEEE, 2019: 1284-1293.
[23] IANDOLA F N, HAN S, MOSKEWICZ M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size[EB/OL]. (2016-11-04)[2022-11-05]. http://arxiv.org/pdf/1602.07360.pdf: arXiv.
[24] DEVLIN J, CHANG M W, LEE K, et al. Bert: pre-training of deep bidirectional transformers for language understanding[EB/OL]. (2019-05-24)[2022-11-05]. http://arxiv.org/pdf/1810.04805.pdf: arXiv.
[25] JIA Z, PADON O, THOMAS J, et al. TASO: optimizing deep learning computation with automatic generation of graph substitutions[C] // Proceedings of the 27th ACM Symposium on Operating Systems Principles. Ontario: ACM, 2019: 47-62.

XIE Xiaoyan, born in 1972. She received the B. S. degree in computer communication from Nanjing University of Science and Technology in 1995. During 1998 – 2002, she stayed in Computer Science Department of Xidian University to study IP based multimedia communication, and received her M. S. degree. She is currently a professor in Xi'an University of Posts & Telecommunications. She is now concentrating on reconfigurable computing and computer vision.