2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP) | 978-1-6654-2701-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASAP52443.2021.00023

Talos: A Weighted Speedup-Aware Device Placement of Deep Learning Models

Yuanjia XU*†‡, Heng WU†, Wenbo ZHANG†‡, Chen YANG*†‡, Yuewen WU† Heran GAO*†‡, Tao WANG†‡
*University of Chinese Academy of Sciences, Beijing, China
†Institute of Software, Chinese Academy of Sciences, Beijing, China
‡State Key Laboratory of Computer Sciences, Institute of Software, Chinese Academy of Sciences, Beijing, China

{xuyuanjia2017,wuheng09,zhangwenbo,yangchen19,wuyuewen11,gaoheran19,wangtao08}@otcaix.iscas.ac.cn

***Abstract*—Efficient device placement of deep learning (DL) models, which consist of many operations, is a big challenge when heterogeneous devices (e.g., CPU, GPU) are considered. Existing average speedup and transient speedup approaches do not make full use of operation-level speedups, and the Total Operation Completion Time (TOCT) cannot be optimized efficiently.**

To address this challenge, we present Talos, a weighted speedup-awareness approach to optimize device placement of multiple DL models. Talos reveals operations within or across DL models have diverse speedups (from 10^{-1} to 10^{2}) on heterogeneous devices. In addition, the execution time of operations are widely ranged (from $0.1ms$ to $100ms$). Talos considers the two features simultaneously as weighted speedups, and treats them as costs in an incremental minimum-cost flow. Compared with state-of-the-art efforts, experiment results show that Talos can reduce TOCT by up to 50%.

***Index Terms*—deep learning models, device placement, heterogeneous devices, minimum-cost flow**

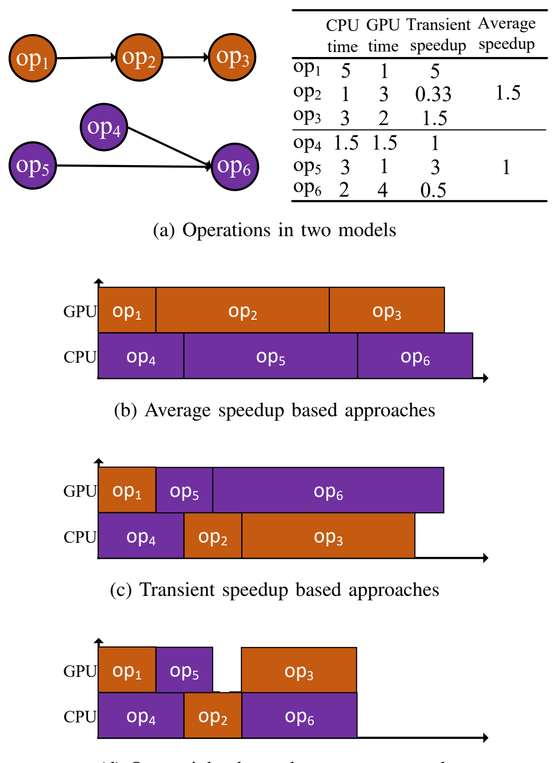
I. Introduction

Deep Learning (DL) models [1] play increasingly important roles in a wide range of application domains, such as image classification and natural language processing. Many heterogeneous devices, such as CPU, GPU, and FPGA, have been leveraged to perform DL models. Sharing devices for multiple DL models is a practical way to improve device utilization since DL models usually cannot fully use allocated devices [2], [3]. Efficient device placement of DL models has attracted many research attentions in recent years [4], [5]. Nowadays, Existing average speedup [2], [6], [7] and transient speedup [8], [9] approaches do not make full use of operation-level speedups, and always ignore the execution time of various operations on heterogeneous devices.

As shown in Fig. 1(a), six operations are in two DL models, op_1, op_2 and op_3 in the first DL model can run sequentially, while op_6 cannot run before both op_4 and op_5 have completed in the second DL model. Here speedup means CPU/GPU ratio and the various speedup features they concern are listed in the right of the table in Fig. 1(a).

(i) Average speedup based approaches [6], [7] calculate speedups from all operations in a DL model. For example, average speedup of op_1 to op_3 is $(5+1+3)/(1+3+2) = 1.5$, so they place op_1, op_2 and op_3 wholly on GPU (Fig. 1(b)),

Corresponding author: zhangwenbo@otcaix.iscas.ac.cn



	CPU time	GPU time	Transient speedup	Average speedup
op₁	5	1	5	
op₂	1	3	0.33	1.5
op₃	3	2	1.5	
op₄	1.5	1.5	1	
op₅	3	1	3	1
op₆	2	4	0.5	

(a) Operations in two models

(b) Average speedup based approaches

(c) Transient speedup based approaches

(d) Our weighted speedup-aware approach

Fig. 1: Device placement results when considering different speedups

since the average speedup of them is larger than $op_4 \sim op_6$'s $(1.5 > 1)$. In this situation, the completion time of op_1 to op_3 and op_4 to op_6 are $1 + 3 + 2 = 6$, $1.5 + 3 + 2 = 6.5$, respectively. And they get TOCT of 12.5 $(6 + 6.5)$.

(ii) Transient speedup based approaches [5], [8] make placement decision of each operation. As shown in Fig. 1(c), when op_6 starts to run, only GPU is available, so they have to place op_6 on GPU and get TOCT of 11.5. Although the TOCT is reduced a little compared with (i), we can easily find that both op_6 and op_3 run with unfit speedups.

So there are huge space of optimization when we consider both the speedup and execution time of each operations in a global view. As illustrated in Fig. 1(d), if we place op_6 on CPU until op_2 finishes, op_6 will get an ideal speedup on CPU while op_3 will also get its ideal speedup running on GPU, then we can get an optimal TOCT of 9 at the cost of a little idle time 0.5 (white blank between op_5 and op_3). Here, op_6 has a speedup of $(0.5 + 2)/4 = 0.625$ in fact, and we call it as a weighted speedup. And we believe that weighted speedup based approach can be more efficient than existing approaches.

However, dealing with speedup features of dozens or even hundreds of operations also introduces a big challenge. In this paper, we present Talos, a flow-based mechanism to optimize device placement of multiple DL models with weighted speedup-awareness. Our contributions include:

- we reveal that homogeneous devices cannot accelerate all operations, and operations' speedups and execution time in different DL models are significantly different. Only considering average or transient speedup to optimize device placement is inefficient (§ II).
- we extend existing homogenous fine-grained sharing primitives to support heterogeneous devices with stable operation speedups, and design an incremental minimum-cost algorithm to optimize the efficiency of device placement (§ III).
- we compare Talos with state-of-the-art approaches, and our experiments with five DL models show that Talos can reduce TOCT by up to 50% while keeping high usage of devices (§ IV).

II. Analysis

In this section, we first formulate the goal of device placement. Then we illustrate our findings of speedup features with experiments on typical DL models (in Fig. 1). After that, we explain why we should introduce a minimum-cost flow and how it can work efficiently with weighted speedup-awareness.

A. Goal Formulation

Given h heterogeneous devices, each with memory capacity hd_i, and M DL models, each of which is a DG (Directed Graph) of operations, then the device placement goal is to place as many operations as possible on devices so that the TOCT is minimized. Table I summarizes key terms used throughout the paper.

TABLE I: **Terms and Notations**. Used in the paper.

h	number of heterogeneous devices
hd_i	memory available per device
M	number of DL models
m_j	number of operations per DL model
$op_{i,j,k}$	the kth operation memory requirement per DL model on ith device
$et(op_{j,k}, hd_i)$	execution time of operation $op_{j,k}$ on ith device
$wt(op_{j,k}, hd_i)$	waiting time of $op_{j,k}$ on ith device
$ct(op_{j,k}, hd_i)$	communication time of $op_{j,k}$ on ith device
$sp(op_{j,k}, hd_i)$	speedup of $op_{j,k}$ on ith device compared to CPU

The goal of device placement is shown in Equation 1. $free(max\ \varepsilon)$ denotes the idle time when we place as many operations as possible ($\sum_{j,k}(op_{i,j,k})/hd_i \le \varepsilon,\ max\ \varepsilon$) on devices. Since operations have different dependencies in DGs, operations may run in parallel on different devices. And we use $lep(M)$ to represent the longest operation execution path in each DL model, so TOCT can be calculated from waiting, execution and communication time of operations in $lep(M)$ for M DL models.

$$\begin{aligned} &min \quad [free(max\ \varepsilon), TOCT] \\ &s.t. \quad \sum_{j,k}(op_{i,j,k})/hd_i \le \varepsilon, \varepsilon < 1, i \in [0,h], \\ &TOCT = \sum_{i,j,k}^{lep(M)} wt(op_{j,k}, hd_i) + et(op_{j,k}, hd_i) + ct(op_{j,k}, hd_i) \end{aligned} \tag{1}$$

B. Findings

We analyse operations' $sp(op_{j,k}, hd_i)$ and $et(op_{j,k}, hd_i)$ (described in Table I) in Tensorflow DL models [1], with the following settings in Table II on a machine with one Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz CPU and two GeForce RTX 2080 Ti GPUs.

TABLE II: Settings of DL Models.

Setting Name	Value
CPU inter-/intra-op threads	10
Batch Size	64
Input Size	255 × 255 × 3
Dataset	ILSVRC-2012

Finding 1: GPU cannot accelerate all operations, thus homogeneous device placement for all operations in a DL model according to average speedups may lead to longer execution time for some operations. As shown in Fig. 2, although the average speedup is near 10 (the red horizontal line), speedups of different operations are in a large range (10^{-1} to 10^2). What's more, there're nearly 30% operations with speedups less than 1 (below the black line in Fig. 2). It means that some operations run slower on GPUs than on CPU. Average speedup based approaches [6], [7] exclusively placing a model on GPUs may probably get suboptimal TOCT.

Finding 2: operation speedups and execution time in multiple DL models are significantly diverse, thus heterogeneous device placement according to transit speedups may interfere high speedup operations in other DL models. As shown in Fig. 3, distributions of operation speedups and execution time in AlexNet, Bert, GoogleNet, ResNet50 and SqueezeNet are significantly different. For example, an operation (Fig. 3(b)) consumes more than 30% execution time in Bert. If it runs on devices with low-speedups in a long period, it may block subsequent operations with high speedups, and increase the TOCT. Similar operations can also be found in other models like ResNet and SqueezeNet. This finding means that speedup settings of op_2, op_6 in our motivating example (Fig. 1) may frequently occur. In these scenarios, transient speedup based approaches may get suboptimal TOCT and longer idle time.

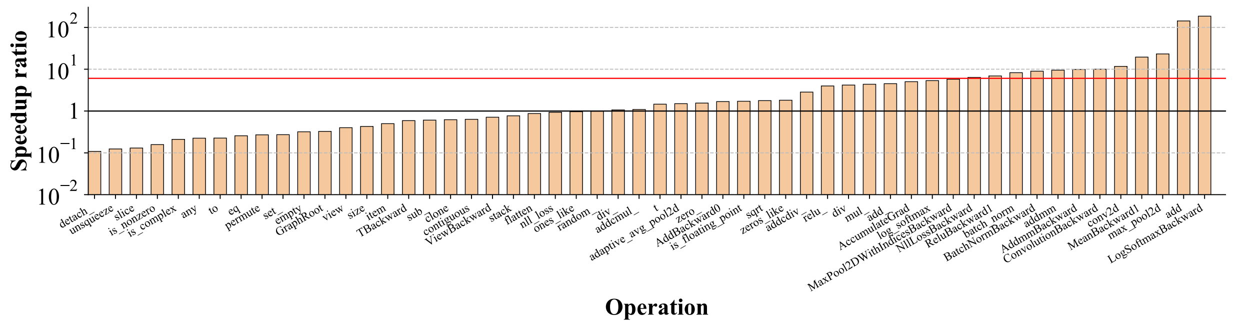


Fig. 2: Speedups for ResNet50 operations running on GPUs

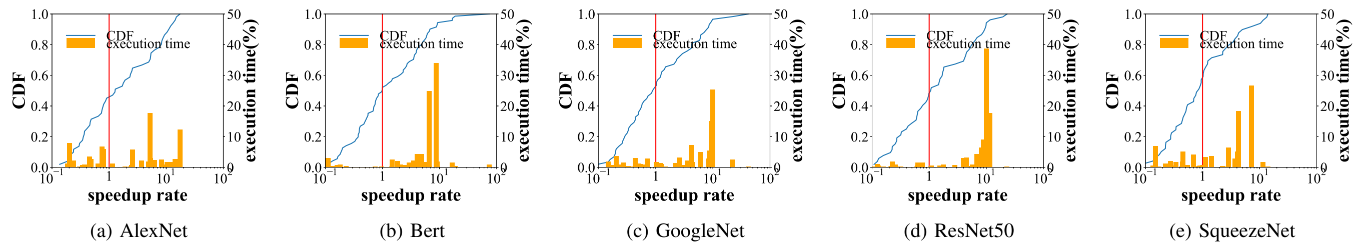


Fig. 3: Distributions of operation speedups and GPU completion time for 5 typical DL models

C. Problem Analysis

Heterogeneous device placement of multiple DL models needs to consider many complex factors affecting TOCT and idle time (e.g., communication time, device capacities, dependencies in DG). In addition, operations continue arriving based on their dependencies, and this continuity can be treated as an incremental flow problem, which has been widely proved in cluster scheduling [6], [10], [11].

A minimum-cost flow graph can be represented by $G = (V, E)$. Here, V denotes vertices of models $model_j$, operations $op_{j,k}$ and devices hd_i (CPU and GPU), respectively. And E denotes edges of $model_j \rightarrow op_{j,k}$ and $op_{j,k} \rightarrow hd_i$. Each edge has a pair of $(capacity, cost)$. As the example shown in Fig. 4, edge $op_2 \rightarrow CPU$ has $(1, 1.5)$ and it means op_2's resource requirement on CPU is 1 (the capacity value in an edge) and placing op_2 on CPU may increase TOCT by 1.5 (the cost value in an edge). A flow $model_1 \rightarrow op_1 \rightarrow GPU$ means we place op_1 on GPU and now TOCT= 1. If we can find all flows with a minimum overall cost, we can place all operations in the graph with minimized TOCT. The ability of minimum-cost flow graph makes it easy for us to support operation weighted speedups on heterogeneous devices:

(i) according to **Finding 1**, it is necessary for our approach to share heterogeneous devices in order to accelerate all operations. To do this, we just need to link device vertices to operation vertices in different DL models by adding new edges. For example, $op_3 \rightarrow CPU$ and $op_3 \rightarrow GPU$ edges in Fig. 4 mean we can place op_3 on CPU or GPU when necessary. If we want to place op_1 on CPU, we just need to add $op_1 \rightarrow CPU$ edge.

(ii) according to **Finding 2**, we need to set operation weighted speedups when sharing heterogeneous devices. For a minimum-cost flow graph, weighted speedups can be represented straightforwardly by cost values on some edges when considering many complex factors. As the example shown in Fig. 4, placing op_3 on GPU, op_6 on CPU can get a minimum cost ($0.5 + 2.5 = 3$ in red dotted lines), and it denotes smaller TOCT and idle time with weighted speedups, otherwise the cost ($4 + 2 = 6$ in black dotted lines) is high and it may denote larger TOCT with existing transient speedups.

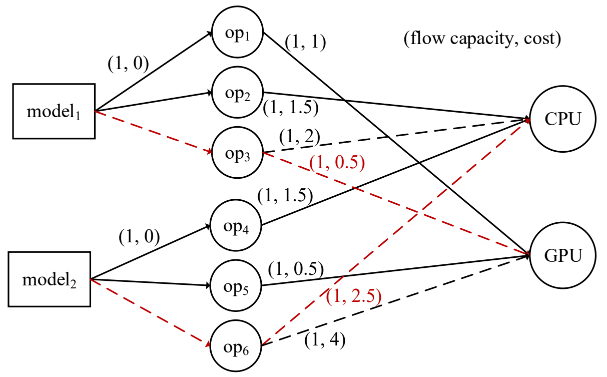


Fig. 4: A minimum-cost flow graph efficiently supporting device placement

III. Design

In order to optimize device placement, Talos' design needs to achieve our device placement goal through the following three aspects:

- How to get stable speedups for operations on heterogeneous devices? (§ III-A)
- How to construct an incremental minimum-cost flow graph for many operations and heterogeneous devices? (§ III-B)
- How to set flow capacity and cost in the graph to support weighted speedup, and solve the minimum-cost flow problem? (§ III-C)

A. Getting stable speedups

Different DL training settings like inter and intra-op threads, batch size and datasets may make operation speedups frequently change. It is necessary to stabilize them with certain settings. Here, the term "stable speedups" is decided by the setting when each operation has the least execution time on any device. In addition, how to find settings with least execution time of operations on shared devices are studied in previous approaches, and we use them to get stable speedups:

- **Inter and intra-op threads.** J. Liu el.al. [12] uses a hill climbing algorithm to find the exact number of inter and intra-op threads that can minimize operation execution time. We run this algorithm to get least execution time when multiple operations arrive on CPU.
- **Memory allocation.** Salus' fine-grained sharing primitive [2] can avoid memory over-consumption and execution interferences of operations on shared GPUs. We use this primitive to get least execution time when multiple operations arrive on GPU.
- **Hyper parameters.** Such parameters including batch size, input size, learning rate and model size of different DL models have official recommendations [1]. We adopt them to run DL models in our environment.

B. Constructing a Minimum-Cost Flow Graph

Maximizing device usages for affinitive operations. Our construction method works as follows. It first finds which operations can be simultaneously placed to maximize ε. Intuitively, operations with diverse speedups tend to use different heterogeneous devices and get less TOCT. Talos uses the term **affinity between operations** to describe this diversity. $dis_{a,b}$ in Equation 2 denotes the diversity of operations in $model_a$ and $model_b$. A higher $Affi(a, b)$ in Equation 3 means a larger diversity and fewer operations in common. $common_{a,b}$ and $total_{a,b}$ in Equation 3 can be identified by operation names (e.g., conv2D, mat, add) in DL models.

$$dis_{a,b} = \sum_{k} \|sp(op_{a,k}) - sp(op_{b,k})\|, k \leq min(C_a, C_b) \quad (2)$$

$$Affi(a, b) = \frac{dis_{a,b}}{common_{a,b}/total_{a,b}} \quad (3)$$

After finding affinitive operations, Talos uses a topological sort algorithm [13] to find out operation dependencies. The operation can be placed only if all of an operation's dependencies are satisfied. Topologies of three typical models are shown in Fig. 5. It first creates vertices for those operations without any in-degree dependency (operations in step1 of Fig. 5) since they can execute immediately. After that, edges between operation vertices and their models are also added to the flow graph.

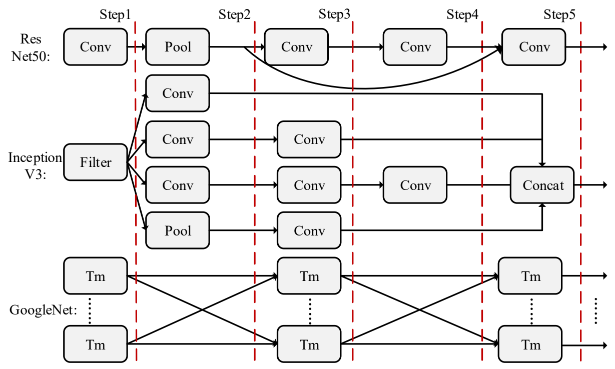


Fig. 5: Finding operations with topological sort

Incrementally updating minimum-cost flow graph. As shown in Fig. 6, by creating new operation vertices and edges, Talos can continue placing operations in following steps (Fig. 5) until all operations are added to it. Since operations in each step can run independently, we can use typical minimum-cost flow algorithm to solve the problem in an incremental manner. In addition, each step only has a few number of operations, so the complexity of the problem can be reduced.

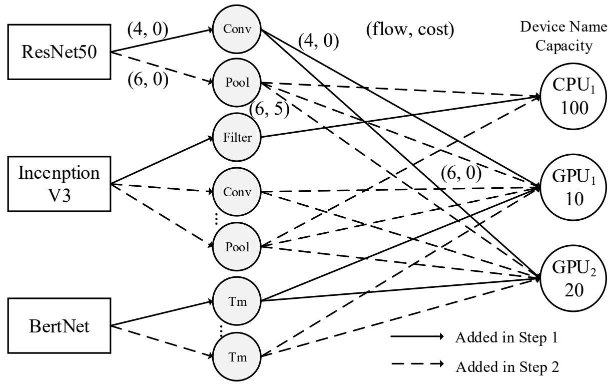


Fig. 6: A flow graph of operations and devices supporting incremental updating

C. Solving the Minimum-Cost Problem

Setting flow capacity to share heterogeneous devices for various operations. Talos first extends existing profiling tools [2] to determine each operation's resource usage $r(op_{j,k}, hd_i)$ (Equation 1) on heterogeneous devices. Then it sets flow capacity values of $model_j \rightarrow op_{j,k} \rightarrow hd_i$ edges. For example in

Algorithm 1 Calculating complex factors to support weighted operation speedup

```
Input:
    $model_j$: a DL model.
    $op_{j,k}$: an operation.
    $hd_i$: a device.
    $et(op_{j,k}, hd_i)$: execution time of $op_{j,k}$ when it runs on $hd_i$
    $currT$: a time stamp when $op_{j,k}$ is ready to run.
 1: for $op_{j,x}$ in $model_j$ do do
 2:   if $op_{j,k}$ $hasDependency$ $op_{j,x}$ then
 3:     $op_{j,x}$ $is$ $on$ $hd_i$ ? $co(op_{j,k}, hd_i)$+=0 : $co(op_{j,k}, hd_i)$
        = max(transferTime($op_{j,x}$, $op_{j,k}$))
 4:   end if
 5:   if $op_{j,k}$ $runPrallel$ $op_{j,x}$ then
 6:     $maxPt(op_{j,k})$ = max(et($op_{j,x}$, $HD$))
 7:   end if
 8: end for
 9: for $op_{j,y}$ on $hd_i$ do do
10:   $dependenceFinishTime$ += ct($op_{j,y}$, $hd_i$)
11: end for
12: If $dependenceFinishTime \leq currT$ $wt(op_{j,k}, hd_i) = 0$
13: Else $wt(op_{j,k}, hd_i) = dependenceFinishTime - currT$
14: if $wt(op_{j,k}, hd_i)$ + $ct(op_{j,k}, hd_i)$ + $et(op_{j,k}, hd_i)$ >
    $maxPt(op_{j,k})$ then
15:   Return $wt(op_{j,k}, hd_i)$,$ct(op_{j,k}, hd_i)$,$maxtPt(op_{j,k}, hd_i)$
16: end if
17: $et(op_{j,k}, hd_i) = 0$, Return 0,0,0
```

Fig. 6, the flow capacity value 4 on edges $ResNet50 \to Conv$ and $Conv \to GPU_1$ means $Conv$ needs 4 resource units of GPU devices. Concurrently, it sets resource capacity to each device vertex. The vertex GPU_1 in Fig. 6 can be shared by $Conv$ operations in $ResNet50$ and $InceptionV3$ since GPU_1's flow sum 10 is equal to GPU_1's capacity 10. Other devices like CPU can be shared in the same way.

Setting costs to support weighted speedups in multiple DL models. After setting flow values, Talos sets each cost on $model_j \to op_k \to device_i$ edges by using Equation 4 in each step (Fig. 5). Here, $cost(op_{j,k}, hd_i)$ denotes $op_{j,k}$'s completion time on hd_i according to above mentioned factors (Section § II-C), and weighted speedup for an operation can be calculated from $cost(op_{j,k}, cpu)/cost(op_{j,k}, hd_i)$. $maxPt(op_{j,k})$ denotes the maximum execution time of operations that can run in parallel with $op_{j,k}$ in a DL model. $wt(op_{j,k}, hd_i)$, $ct(op_{j,k}, hd_i)$ and $et(op_{j,k}, hd_i)$ are explained in Table I. If placing operation $op_{j,k}$ on device hd_i can reduce more time, $cost(op_{j,k}, hd_i)$ has a less value (a larger weighted speedup). Distinct meanings of these variables are shown in Fig. 7.

$$\begin{aligned} & cost(op_{j,k}, hd_i) \\ & = wt(op_{j,k}, hd_i) + ct(op_{j,k}, hd_i) + et(op_{j,k}, hd_i) \\ & - maxPt(op_{j,k}) \end{aligned} \tag{4}$$

Here, Talos gets $et(op_{j,k}, hd_i)$ by profiling operations on different devices. $wt(op_{j,k}, hd_i)$, $ct(op_{j,k}, hd_i)$ and

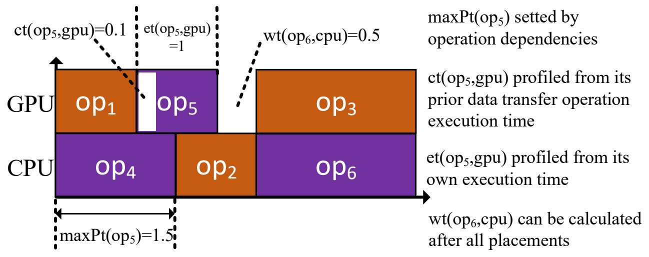

ct(op₅,gpu)=0.1
et(op₅,gpu) =1
wt(op₆,cpu)=0.5
maxPt(op₅) setted by operation dependencies
GPU
CPU
op₁
op₅
op₃
op₄
op₂
op₆
ct(op₅,gpu) profiled from its prior data transfer operation execution time
et(op₅,gpu) profiled from its own execution time
maxPt(op₅)=1.5
wt(op₆,cpu) can be calculated after all placements

Fig. 7: Distinct meanings of time variables

$maxPt(op_{j,k})$ can be calculated in Algorithm 1. Algorithm 1 traverses a DL model and finds $op_{j,k}$'s dependent and parallel operations (Lines 1-8), respectively. Concurrently it can calculate $co(op_{j,k}, hd_i)$ and $maxPt(op_{j,k})$. Then it traverses all operations in $model_j$ running on hd_i and calculates $wt(op_{j,k})$ (Lines 9-12). If $op_{j,k}$ is not in the longest execution path and Talos just sets $cost(op_{j,k}, hd_i)$ to 0. Otherwise it sets $cost(op_{j,k}, hd_i)$ with a positive value (Lines 13-17).

Minimizing TOCT with an incremental Minimum-Cost FLow (MCF) algorithm. Talos can always get an incremental flow graph in each step (Fig. 5), and incrementally uses a typical MCF algorithm to minimize TOCT. To reduce the number of edges and vertices, adjacent operations with similar speedups and execution time can be packed together. Here, we use the term **speedup variance** sv and **execution time variance** tv to decide whether to pack operations. For a set of adjacent operations, if the first operation in the set has the speedup 2 and sv = $(0.5, 2)$, tv=10ms, we can pack operations having speedups in the range of $(0.5, 2) \times 2 = (1, 4)$ and execution time less than 10ms together.

In our scenario, flow in each edge should be equal to its capacity (an operation placed on a device) or keep zero (an operation waiting to be placed), and existing scaling based MCF algorithms [13] may not work well for device placement. Therefore, Talos uses successive shortest path [13] to solve the problem. The complexity of our incremental algorithm is $O(h \times C_{step} \times (A + C_{step} + M)^2)$, where h, C_{step}, M denote the number of devices, the average count of operations in each step (Fig. 5) and the number of models, respectively.

D. Discussion

Extending affinity definition. Motivated by the definition in Aladdin [11], affinity between operations is calculated from two DL models for simplicity. When multiple DL models arrive, it can be extended to support more than two DL models by changing the distance in Equation 2.

Keeping speedup stable. Many settings affect speedups, and we use those settings with least operation execution time to get stable speedups in an offline manner [2], [12]. Thus settings are already known before DL training. We leave the change of online operation speedups (caused by input size, available devices, etc.) to further work.

Supporting more heterogeneous devices. Due to GPUs are widely supported and optimized by Tensorflow [14] and Pytorch [15], we can get stable operation speedups of CPU/GPU. Currently, placing operations on FPGAs, Google TPUs still needs a lot of manual optimizations and open-sourced implementations. Talos' speedup-aware approach can support more devices by reusing device-specific optimizations.

Determining efficiency metrics. TOCT and idle time are the two metrics in Talos (Equationg 1) to evaluate the efficiency. Both of them are profiled from computing resources on heterogeneous devices, while memory efficiency has been well studied in [2], [3].

IV. Evaluation

We now evaluate how well Talos optimizes device placement of multiple DL models:

1) can Talos get reduced idle time of heterogeneous devices?
2) how well is the optimization of TOCT when using Talos compared with existing approaches?
3) what is the overhead of Talos and is it acceptable?

A. Experimental Settings

DL models. We use five DL models: AlexNet, GoogleNet, ResNet-50, Bert, and SqueezeNet [1]. Their datasets include CIFAR-10, ImageNet-2012 and CoLA. Batch sizes of AlexNet, GoogleNet, ResNet-50 and SqueezeNet are set to 64, while batch size of Bert is 32 according to Tensorflow recommendation.

Heterogeneous Devices. We use one Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz CPU (24 cores, 128GB RAM) and two NVIDIA GeForce RTX 2080 Ti GPUs (4352 cuda cores, 11 GB RAM) on a machine as heterogeneous devices. By recording operation traces on the machine, complex speedup factors on heterogeneous resources can be calculated and compared. All settings are based on Tensorflow 1.14 and have been tuned (described in Section § III-A) to get stable operation speedups.

Speedup Variance Threshold. We set $sv = (0.5, 2)$, tv=10ms (described in Section § III-C(iii)) as default.

Environment. We compare minimized idle time and TOCT of the following approaches:

- **ASWM** (average speedup within a DL model based approaches [2], [6], [7]): these approaches place all operations in a DL model on GPU when they have larger average speedups.
- **TSAM** (transient speedup across DL models based approaches [8], [9]): these approaches place operations on available devices with largest speedups.
- **Talos** (our approach): Talos uses a minimum-cost flow algorithm to optimize device placement.

B. Reduced Idle Time

Metrics. We use **idle time** to denote how long the computing resources in heterogeneous devices are wasted. When a placement result leads to less idle time, Talos can get higher device usage and it can continue to minimize TOCT.

Fig. 8 shows the comparison result. Talos outperforms TSAM for all operations and reduces idle time in the range from 16.6% to 93.3%. Instead of an single operation waiting to run only on one device, Talos can reduce more waiting time $wt(op_{j,k}, hd_i)$ by simultaneously using heterogeneous devices. And we believe it is the reason why Talos has less idle time when DL models arrive. In addition, changing sv and tv (different operation numbers in same TOCT) may also affect idle time. Our experiment in Fig. 8 is set to $sv = (0.5, 2), tv = 10$ and it can reduce the most idle time. Talos' less idle time in operations in five DL models reveals that it can help to further minimize TOCT while keeping high device usages.

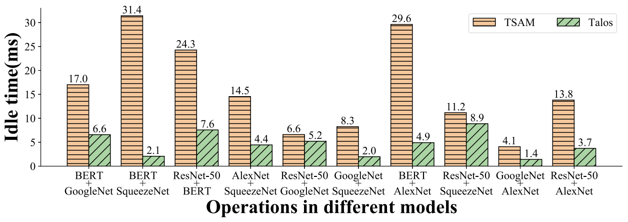


Fig. 8: Idle time when reassigning operations.

TABLE III: The affinity values between operations in any two models.

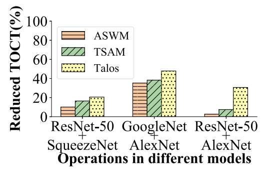
Two models	Affinity value
Bert+GoogleNet	0.14
Bert+Squeeze	0.17
ResNet50+Bert	0.20
AlexNet+SqueezeNet	0.29
ResNet50+GoogleNet	0.21
GoogleNet+SqueezeNet	0.32
Bert+AlexNet	0.37
ResNet50+SqueezeNet	0.46
GoogleNet+AlexNet	0.53
RestNet50+AlexNet	0.56

C. Reduced TOCT

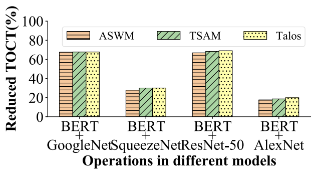
Metrics. We calculate values of $Affi(a, b)$ (Equation 3) between operations in any two DL model combination and sort them in an $Affi(a, b)$ ascending order. Then we compare reduced TOCT percentages after placing operations with high affinities. We place operations having $Affi(a, b) > 0.4$ by empirical studies.

We calculate the affinities (with sv = $(0.5, 2)$, tv = 10ms) between operations in any two models (Equation 3) and they are shown in Table III.

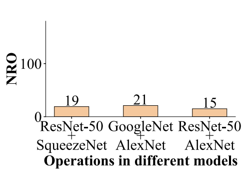
As shown in Fig. 9 (a), We compare reduced TOCT when $Affi(a, b) > 0.4$. Talos works well on four models (ResNet50, SqueezeNet, GoogleNet, AlexNet) in three combinations, and outperforms ASWM and TSAM in a range from 12% to 300% (31%/9.7% when ResNet50+AlexNet). Talos reduces TOCT by up to 50% in GoogleNet+AlexNet. Such improvements only need to reassign a small number of operations. This result indicates that Talos can place high



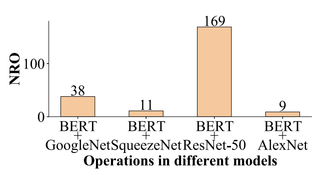
(a) TOCT with operation affinities > 0.4



(b) TOCT with operation affinities < 0.4

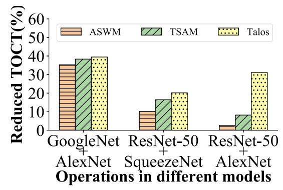


(c) Reassigned operation number with affinities > 0.4

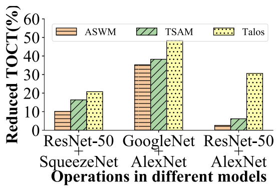


(d) Reassigned operation number with affinities < 0.4

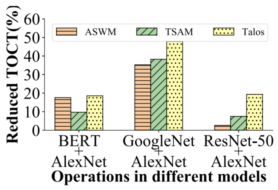
Fig. 9: Reduced TOCT and the number of reassigned operations with different approaches



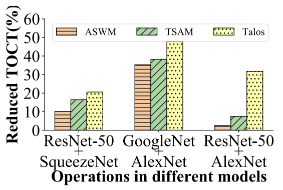
(a) sv=(0.5,2), tv=5ms



(b) sv=(0.5,2), tv=20ms



(c) sv=(0.3,3.3), tv=10ms



(d) sv=(0.7,1.43), tv=10ms

Fig. 10: Reduced TOCT for affinitive operations with different speedup variance thresholds

speedup operations on GPU while low ones are on CPU. Such placements can reduce more TOCT than ASWM and TSAM.

Apart from above four models, Talos has almost the same TOCT comparing with ASWM and TSAM when placing operations in Bert (Fig. 9 (b)) when $Affi(a,b) < 0.4$. Reduced percentages are in the range from 23.7% to 71%. The reason why operations in these DL models are not affinitive is that there are two long operations in Bert (Fig. 3(b)) consuming more than half completion time. Note that such imbalance can be alleviated by DL deployment tools like TVM [16] and NNFusion [17], and we believe Talos' weighted based speedup-aware mechanism can reduce more TOCT after DL model modification by deployment tools.

Fig. 10(a) and Fig. 10(b) show the change of reduced TOCT for affinitive operations with different tv. When tv becomes larger, the affinity value changes slightly. Operations in GoogleNet+AlexNet becomes more affinitive than those in RestNet-50+SqueezeNet and Talos reduces more TOCT. We think most of operation execution time in GoogleNet+AlexNet is between 5ms and 20ms, thus it can make Talos perform better. Changing of tv does not affect other two DL models since their execution time is mostly larger than 20ms.

Fig. 10(c) and Fig. 10(d) show the affect of changing sv. When sv is in a smaller range, we will get more operations and the speedup variance between packed operations is enlarged. Thus Talos may reassign more operations and reduce more TOCT (ResNet-50+AlexNet). If operation speedup range is not in $(0.3, 3.3)$ or $(0.7, 1.43)$, changing sv will not change placement results (GoogleNet+AlexNet). This means that if a smaller range of sv reduces more TOCT, we can continue to narrow sv until TOCT is minimized.

D. Overhead of Device Placement

Metrics. The overhead of device placement can be evaluated from the following two metrics.

- **NOD** (the number of packed operations with different sv and tv): a DL model consists of many operations and runs "forward-backward" computing in only a few seconds. If we cannot control NOD, more communication time $ct(op_{j,k}, hd_i)$ that could increase TOCT.
- **NRO** (the number of reassigned operations): Talos may reassign operations on other devices to reduce more TOCT. However, a large NRO means increased communication time $ct(op_{j,k}, hd_i)$ between operations on different devices. It can also affect TOCT.

Environment. We compare NOD under six different thresholds in two setting groups: 1) a group based on changing sv: $sv = (0.3, 3.33), tv = 10$, $sv = (0.5, 2), tv = 10$, $sv = (0.7, 1.43), tv = 10$. 2) a group based on changing tv: $sv = (0.5, 2), tv = 5$, $sv = (0.5, 2), tv = 10$, $sv = (0.5, 2), tv = 20$).

TABLE IV: **NOD:** the number of packed operations with different sv and tv

sv, tv	AlexNet	Google Net	ResNet -50	Squeeze Net	BERT
$(0.3, 3.33), 10$	23	94	92	19	195
$(0.5, 2), 10$	27	114	129	42	204
$(0.7, 1.43), 10$	29	117	180	43	204
$(0.5, 2), 5$	38	155	153	55	326
$(0.5, 2), 20$	19	79	107	29	142

As shown in Table IV, NOD is in a range from 19 to 326 for five DL models. When sv and tv have larger values, NOD becomes smaller accordingly. Same trends can be drawn when sv and tv get less values. Comparing to the operation number in a DL model, NOD is $10\times$ to $100\times$ reduced and this makes

Talos be able to finish in a short time while communication time can be reduced to the same trend.

Fig. 9(c) and Fig. 9(d) indicates that reducing most TOCT does not need to reassign too many affinitive operations. Talos can reduce TOCT only by reassigning a few (15-21) bottleneck operations. NRO is extremely small, such a number of operations have their communication time in less than 1% of TOCT, which is acceptable is our scenario.

V. RelatedWork

Transient speedup across DL models based approaches. Recent researches use systemic and algorithmic mechanisms to get optimized device placement results. Mirhoseini et al. [5] train a reinforcement learning method to place operation for typical DL models. Liu et al. [12] extend OpenCL programming model to support the heterogeneous devices [18], and use a hill climbing algorithm to maximize speedups of operations on CPU. Salus [2] and μLayer [19] assume that all operations have same speedups and split DL models on GPUs to improve device efficiency.

Talos locates that the bottleneck of device placement is caused by diverse operation speedups, and it uses weighted speedups, which is different from existing approaches.

Average speedup within a DL model based approaches. These approaches apply heuristics to place operations to support the heterogeneity of devices. Tiresias [20] and Aladdin [11] propose priority and affinity setting methods to reduce completion time of DL models. Gandiva [3] observes the cyclical memory usage pattern of DL models and the pattern can be used to minimize immigration time and memory contentions [2]. Gavel [7] and Allox [6] redefine fairness and performance by average speedup.

We also do a deep analysis to disclose speedup and distribution of operations in DL models, and Talos' design suggests the feasibility of flow graph in device placement.

Performance profiling for DL models. Profilers can gather detailed performance data across different DL systems. Enterprises including Google [21] ignite the performance improvement gained from heterogeneous devices, and the importance of device placement on enterprise shared clusters. Clockwork [22] profiles DL models through many perspectives including model size, hardware type, OS kernel and application environment. Operation implementations [16], [17] and device concurrent threads [12] can also affect the performance and their weighted speedups.

Existing performance profiling approaches indicate that many hardware and software settings can make operation performance fluctuations, and we extend some of existing approaches to stabilize operation speedups.

VI. Conclusion

We presented Talos, a weighted speedup-aware device placement approach. Talos reveals the reason why existing approaches are inefficient, and uses an incremental flow-based mechanism to optimize device placement. Experiments with five DL models demonstrate that Talos can improve device placement efficiency by minimizing TOCT and idle time. Our further work includes: (i) we will extend Talos to support new DL models on more devices in an online manner, since Talos now profiles operations and get DL settings offline. (ii) we will further test Talos with deployment optimization (e.g., TVM [16], NNFusion [17]) and reduce more TOCT, since sometimes affinity of native operations could be low without deployment optimization in Tensorflow [14] and Pytorch [15].

Acknowledgement. This work is supported in part by National Key R&D Program of China (No. 2018YFB1402503), National Natural Science Foundation of China (No. 61872344) and Youth Innovation Promotion Association of Chinese Academy of Sciences Fund (No. 2018144).

References

[1] "Tensorflow models," https://github.com/tensorflow/models.

[2] P. Yu and M. Chowdhury, "Salus: Fine-grained gpu sharing primitives for deep learning applications," *arXiv preprint arXiv:1902.04610*, 2019.

[3] W. Xiao, R. Bhardwaj, R. Ramjee, M. Sivathanu, N. Kwatra, Z. Han, P. Patel, X. Peng, H. Zhao, Q. Zhang *et al.*, "Gandiva: Introspective cluster scheduling for deep learning," in *OSDI*, 2018, pp. 595–610.

[4] T.-A. Yeh, H.-H. Chen, and J. Chou, "Kubeshare: A framework to manage gpus as first-class and shared resources in container cloud," in *HPDC*, 2020, pp. 173–184.

[5] Y. Gao, L. Chen, and B. Li, "Spotlight: Optimizing device placement for training deep neural networks," in *ICML*, 2018, pp. 1676–1684.

[6] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: compute allocation in hybrid clusters," in *Eurosys*, 2020, pp. 1–16.

[7] D. Narayanan, K. Santhanam, F. Kazhamiaka, A. Phanishayee, and M. Zaharia, "Heterogeneity-aware cluster scheduling policies for deep learning workloads," in *OSDI*, 2020, pp. 481–498.

[8] B. Jeon, L. Cai, P. Srivastava, J. Jiang, X. Ke, Y. Meng, C. Xie, and I. Gupta, "Baechi: fast device placement of machine learning graphs," in *SoCC*, 2020, pp. 416–430.

[9] X. Yi, Z. Luo, C. Meng, M. Wang, G. Long, C. Wu, J. Yang, and W. Lin, "Fast training of deep learning models over multiple gpus," in *Middleware*, 2020, pp. 105–118.

[10] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale," in *OSDI*, 2016, pp. 99–115.

[11] H. Wu, W. Zhang, Y. Xu, H. Xiang, T. Huang, H. Ding, and Z. Zhang, "Aladdin: Optimized maximum flow management for shared production clusters," in *2019 IEEE IPDPS*, May 2019, pp. 696–707.

[12] J. Liu, D. Li, G. Kestor, and J. Vetter, "Runtime concurrency control and operation scheduling for high performance neural network training," in *IPDPS*. IEEE, 2019, pp. 188–199.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[14] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016, pp. 265–283.

[15] "pytorch1.6," https://github.com/pytorch/pytorch/tree/1.6.

[16] "Tvm," https://github.com/apache/tvm.

[17] "Nnfusion," https://github.com/microsoft/nnfusion.

[18] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-memory for energy-efficient neural network training: A heterogeneous approach," in *2018 MICRO*. IEEE, 2018, pp. 655–668.

[19] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "μlayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Eurosys*, 2019, pp. 1–15.

[20] J. Gu, M. Chowdhury, K. G. Shin, Y. Zhu, M. Jeon, J. Qian, H. Liu, and C. Guo, "Tiresias: A gpu cluster manager for distributed deep learning," in *16th USENIX NSDI 19*, 2019, pp. 485–500.

[21] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the next generation," in *Eurosys*, 2020, pp. 1–14.

[22] A. Gujarati, R. Karimi, S. Alzayat, A. Kaufmann, Y. Vigfusson, and J. Mace, "Serving dnns like clockwork: Performance predictability from the bottom up," *arXiv preprint arXiv:2006.02464*, 2020.