SwapAdvisor。通过智能交换推动深度学习超越GPU内存极限

黄建钦
纽约大学

顾瑾
纽约大学

李金阳
纽约大学

摘要

众所周知，更深更广的神经网络可以达到更好的精度。但由于GPU内存有限，增加模型大小的趋势很难继续。一个有希望的解决方案是支持GPU和CPU内存之间的互换。然而，现有的交换工作只能处理某些模型，不能达到令人满意的性能。

深度学习的计算通常表示为数据流图，可以通过分析来改善交换。我们提出了SwapAdvisor，它基于给定的数据流图在三个方面进行联合优化：操作员调度、内存分配和交换决策。SwapAdvisor使用一个定制设计的遗传算法探索广阔的搜索空间。使用各种大型模型进行的评估表明，SwapAdvisor可以训练高达12倍于GPU内存限制的模型，同时实现53-99%的假设基线与无限GPU内存的吞吐量。

ACM参考格式。
黄建钦，顾瑾，和李景阳。2020.SwapAdvisor:通过智能交换推动深度学习超越GPU内存极限。载于*第二十五届国际编程语言和操作系统架构支持会议（ASPLOS '20）论文集，2020年3月16-20日，瑞士洛桑*。ACM，New York，NY，USA，15 pages. https://doi.org/10.1145/ 3373376.3378530

1　简介

在过去的几年里，深度学习社区一直在使用更大的深度神经网络（DNN）模型，以便在更复杂的任务上获得更高的准确性[16, 47, 55, 58, 60]。经验证据表明，自80年代以来，由于硬件的改进和大型数据集的可用性，最先进的神经网络的参数数量大约每2.4年就会翻一番[10]。然而，今天可以探索的DNN模型的规模为受限于有限的GPU内存（例如，NVIDIA V100 GPU的16GB）。

现有的工作试图通过使用较低精度的浮点[12, 24]来减少内存消耗，或者通过量化和拼接来压缩模型参数[3, 9, 13-15, 20]。然而，这些技术会影响模型的准确性，并需要进行大量的超参数调整。其他一些解决方案会丢弃中间数据，在需要时再丢弃[2, 11, 29]。然而。重新计算它们不能支持大型模型，因为模型参数不容易被重新计算。

在不影响精度的情况下，解决GPU内存限制的一个有前途的方法是在DNN计算过程中在GPU和CPU内存之间交换张量数据[26, 30, 40, 49]。有几个技术趋势使交换成为可能：1）CPU内存比GPU内存大得多，也便宜得多；2）现代GPU硬件可以有效地将通信与计算重叠；3）GPU和CPU之间的通信带宽现在已经足够好，而且随着PCIe 5.0[37]的到来和NVLink[26, 36]的广泛采用，通信带宽将大幅增长。

DNN计算的交换与传统的交换（在CPU内存和磁盘之间）不同，因为DNN的计算结构通常在执行前就已经知道了。例如，以数据流图的形式。这样的知识释放出巨大的机会，通过最大限度地叠加计算和通信来优化交换的性能。不幸的是，现有的工作要么不利用这些信息（例如TensorFlow的交换扩展[57]），要么只是基于手动判例[26, 30, 49]以初级的方式使用它。例如，TFLMS[26]和vDNN[40]只根据图中的拓扑排序来交换激活张量。SuperNeurons[49]只交换卷积操作的数据。因此，这些工作不仅只支持有限类型的DNNs，而且也不能实现交换的全部性能潜力。

在本文中，我们提出了SwapAdvisor，一个通用的交换----------。ping系统，可以在有限的GPU内存中支持各种大型模型的训练和推理。对于一个给定的DNN计算，SwapAdvisor在执行之前*精确地*计划了交换的内容和时间，以最大限度地提高计算和通信的重叠。

仅仅是数据流图还不足以实现这种精确的规划，这还取决于如何安排运算符的执行以及如何进行内存分配。更重要的是，内存分配和运算符

ASPLOS '20，2020年3月16-20日，瑞士洛桑。

acm isbn 978-1-4503-7102-5/20/03.. . $15.00
https://doi.org/10.1145/3373376.3378530

调度也严重影响了可实现的最佳交换性能。SwapAdvisor使用一个定制设计的遗传算法来搜索所有的内存分配空间和

操作员调度，因此最终的交换计划代表了对操作员调度、内存分配和交换的联合优化结果。

我们的评估显示，SwapAdvisor可以支持非常大的模型训练，最高可达12倍的GPU内存限制。SwapAdvisor实现的吞吐量是无限GPU内存的假设基线的53%-99%。Swa- pAdvisor也可用于模型推理。推理的内存占用比训练要小。然而，为了节省成本，人们可以让多个模型使用一个GPU。在这种情况下，人们可以使用SwapAdvisor来限制每个模型只使用一小部分内存，而不是在各个模型之间共享整个内存。我们的实验表明，与其他时间共享方法相比，SwapAdvisor实现了高达4倍的低延迟。

就我们所知，SwapAdvisor是第一个支持各种大型DNN模型的通用交换系统。虽然有希望，但SwapAdvisor有几个局限性（第8节）。特别是，它需要一个没有控制流原语的静态数据流图，并且只为单个GPU计划交换。消除这些限制需要进一步研究。

2　背景介绍

DNN的训练和推理通常在GPU上完成，GPU通过高性能总线（如PCIe和NVLink）连接到主机CPU。GPU使用不同的内存

技术，具有更高的带宽，但容量有限，例如，NVIDIA V100上的16GB。相比之下，CPU配备数百GB的内存是很常见的。因此，在GPU和CPU内存之间交换数据是很有吸引力的，以支持训练和推理，否则在GPU内存的限制下是不可能的。

现代DNN已经发展到由多达数百个层组成，这些层通常以一种复杂的非线性拓扑结构组成。TensorFlow/MXNet等编程框架将DNN的计算表达为张量运算的数据流图。DNN的内存消耗分为三类。

1. 模型参数。在DNN训练中，参数在一个迭代结束时被更新，并在下一个迭代中使用。参数张量与DNN模型的 "深度"（层的数量）和 "宽度"（层的大小）成正比。对于大型模型，这些参数在内存使用中占主导地位。
2. 中间结果。这些结果包括激活、梯度和误差张量，其中后两者只在训练中出现，而在推理中没有。
3. 划痕空间。某些运算符的实现（如卷积）需要划痕空间，最高可达一千兆字节。划痕空间是总内存使用量的一小部分。

现有的工作使用基于不同类别的内存使用模式的人工启发式方法。例如，先前的工作不交换参数[1]，但只交换激活到CPU[26, 40]。如果没有参数交换，先前的工作不能支持参数不适合在GPU内存中的DNN。此外，基于人工启发式的设计错过了提高性能的机会，因为现代DNN的数据流图太复杂了，无法由人分析。

在本文中，我们提出了一种通用的交换机制，任何张量都可以在内存压力下被换入/换出。更重要的是，我们的目标是摆脱人工启发式的做法，自动优化给定的任意复杂数据流图的最佳交换计划。我们将讨论的重点放在单个GPU的交换上，但我们的设计可以用于多GPU的训练设置中，通过数据平移在不同GPU上复制模型。

3　挑战和我们的方法

一个好的交换计划应该尽可能地将通信和计算重叠起来。过度重叠的机会来自于换出一个（暂时）未使用的张量，以便在操作者执行需要之前为换入一个超出内存的张量腾出空间。

最大限度地提高这种重叠的

我们的目的 当 在数据流的帮助下，

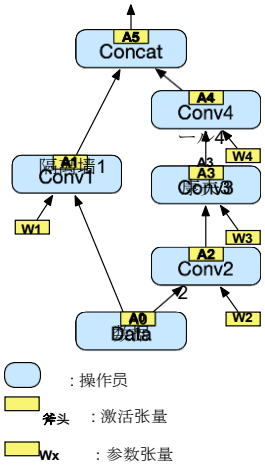
是为了什么 通过仔细的规划来进行交换

和图。

之前的工作试图找到一个好的交换计划heuristi--。仅仅基于数据流图结构[26, 40, 49]。然而，这还不够。特别是，我们认为有两个关键因素影响交换规划。

- *内存分配*。DNN计算使用的张量大小范围很广，从几KB到几百MB。为了提高速度和减少内部碎片，MXNet等框架使用了一个内存池，该内存池在各种大小的类别中预先分配了一些固定大小的张量对象。因此，交换不仅发生在GPU内存满的时候，而且发生在某个特定大小类没有空闲对象的时候。因此，如何配置内存池进行分配会严重影响交换的性能。
- *操作员调度*。现代DNN往往有复杂的数据流图，因为各层不再形成一个链，而是包含分支、连接和未滚动的循环。因此，有许多不同的潜在调度规则来执行操作符。执行的顺序

[1] 唯一的例外是SuperNeuron[49]，它交换了卷积而不是其他类型的参数。



(a) 一个模型的部分数据流图。

时间	t1	t2	t3	t4	t5	t6	t7
计算	数据	Conv1	隔离区2	康夫3	コンボール4	并列	
交换	W1	W2	W3	W4	A1		
互换			A1	A0	A3	A2	
内存使用情况	A0W1	A0A1W1W2	A0A1A2W2W3	A0A2A3W3W4	A1A2A3A4W4	A1A2A4A5	

(b) 首先安排左边的分支。分配5个内存对象，每个都是2MB。

时间	t1	t2	t3	t4	t5	t6	t7
计算	数据	隔离区2	康夫3	コンボール4		Conv1	并列
交换	W2	W3	W4		W1		
互换				A2	A2	A3	A0
内存使用情况	A0W2	A0A2W2W3	A0A2A3W3W4	A0A2A3A4W4	A0A2A3A4W1	A0A1A3A4W1	A0A1A4A5

(c) 首先安排右边的分支。分配五个内存对象，每个都是2MB。

时间	t1	t2	t3	t4	t5	t6	t7
计算	数据	隔离区2	康夫3	コンボール4	Conv1	并列	
交换	W2	W3	W4	W1			
互换				A2	A2	A3	
内存使用情况	A0W2	A0A2W2W3	A0A2A3W3W4	A0A2A3A4W1W4	A0A1A2A3A4W1	A0A1A3A4	

(d) 先安排右边的分支。分配九个内存对象，一个是2MB，其他的是1MB。

时间	t1	t2	t3	t4	t5	t6	t7
计算	数据	Conv1	隔离区2	康夫3	コンボール4		并列
交换	W1	W2	W3	W4			
互换				A0	A2	A2	A3
内存使用情况	A0W1	A0A1W1W2	A0A1A2W2W3	A0A2A3W3W4	A1A2A3A4W4	A1A2A3A4	A1A3A4A5

(e) 先安排左边的分支。分配九个内存对象，其中一个是2MB，其他的是1MB。

除了A2和A5，所有的张量都是1MB。其中2MB。GPU内存为10MB。

图1.一个数据流图的不同时间表和内存分配。

可以深刻地影响内存的使用，从而影响交换的性能。

例子。我们用一个例子来说明内存分配和调度是如何影响交换的。这个例子是基于一个玩具神经网络的数据流图的一部分，如图1（a）所示。为了简单起见，数据流图只显示了前向传播，而省略了后向传播部分。这种分支结构在现代CNN中很常见[47, 60]。

在图1（a）中，蓝色的圆角矩形代表运算符，黄色的小矩形代表张量。一个张量被标记为A_x（激活张量）或W_x（参数张量）。假设所有的张量都是1MB，除了A_2和A_5，它们是2MB（因为A_2、A_5是用来连接两个路径的）。因此，内存消耗为12MB[2]。假设GPU的内存容量为10MB，执行一个运算器或在GPU和CPU之间传输1MB的数据需要一个单位的时间。

一个参数张量最初是在CPU内存中，在使用前必须被交换到GPU内存中。我们可以换出一个参数张量而不把它复制到CPU内存，因为它在前向传递过程中没有改变。相比之下，不需要换入激活张量（因为它是由运算符创建的），但在换出时必须将其复制到CPU内存中，因为在后向传递中需要它。

对于图1(a)，有很多方法可以分配内存和安排执行。我们展示了2个时间表的例子：*左先*执行左边分支上的运算符，*右先*执行右边分支。我们展示了2个内存分配的例子：*粗粒度*分配5个内存对象，每个2MB；*细粒度*分配8个内存对象，每个1MB和1个2MB的对象。总的来说，有4种时间表/分配的组合，我们展示了最佳的交换方式

图1(b)至(e)为每种组合下的计划。由于GPU-CPU的通信是双工的，并且与GPU的执行同时进行，每个表的前三行分别给出了GPU计算、换入（从CPU到GPU）和换出（从GPU到CPU）的行动时间线。最后一行显示了当前驻留在GPU内存中的张量对象。

让我们对比一下图1(c)和(d)，看看为什么内存异构会影响交换计划。(c)和(d)都有相同的右先调度。然而，(c)的总执行时间是

(c)比(d)的时间长一个单位。具体来说，在图1(c)中，GPU在时隙t_5中闲置，而算子$Conv_1$等待其参数W_1被换入。不可能提前换入W_1，因为5个2MB对象的粗粒度内存池在时间t_4时已经满了。我们不能提前换出5个GPU驻留对象中的任何一个。A_3,W_4和A_4是当前运行的运算器$Conv_4$需要的输入/输出张量，而A_0则需要作为下一个运算器$Conv_1$的输入。A_2正在被换出，但由于其尺寸较大，通信需要两个单位的时间。图1使用了一个有八个1MB对象和一个2MB对象的细粒度内存池。因此，它可以提前一个单位时间，在t_4，换入操作者$Conv_1$需要的W_1，因为内存池中还有空间。

我们对比图1(d)和(e)，看看为什么调度的原因影响到交换规划。(d)和(e)都使用相同的细粒度内存池。然而，图1(e)比(d)多花了一个单位的时间，因为它的进度是先左后右。在图1(e)中，GPU在时隙t_6内是空闲的，因为操作者*Concat*等待2MB的张量A_2完成交换，以便为其2MB的输出A_5腾出空间。不可能提前换出A_2，因为它是运算器$Conv_3$的输入，该运算器在时间t_4执行。相比之下，图1(d)的右一时间表能够提前在t_3执行$Conv_3$，从而使A_2提前被换出。

[2] 我们不考虑例子中的内存重用，因为部分数据流图不包括禁止许多重用情况的后向传递。

我们的方法。由于内存分配和操作者调度会严重影响交换的性能，我们假设一个给定的数据流图以及相关的内存分配方案和操作者调度（第4节），得出一个交换计划（即哪些张量要换入/换出以及何时换出）。具体来说，交换计划通过交换未来最长时间内不需要的张量和尽可能早地预取之前交换的张量来优化计算和通信的重叠。

我们搜索可能的内存分配和运算器调度的空间，以找到一个具有最佳交换性能的组合。我们采用遗传算法（简称GA）[5, 8, 18]来搜索内存分配和操作者调度的良好组合，而不是使用人工启发式方法来约束和指导搜索。遗传算法已经被用于NP-hard组合问题[28, 39]和并行系统的调度[43]。在其他搜索启发式方法（如模拟退火）中，我们选择了GA，因为它是快速的。GA可以在多核CPU上进行并行化和高效计算。

为了能够有效地探索庞大的搜索空间，我们必须能够快速评估交换计划在任何内存分配/调度组合下的整体性能（即端到端的执行时间）。我们发现在真正的框架上进行实际执行太慢了。因此，我们通过在数据流引擎模拟器下运行交换计划来估计性能。该模拟器使用测量的每个操作者的计算时间以及GPU-CPU的通信带宽，因此它可以估计在给定的调度、内存分配和交换计划下数据流图的执行时间。我们的模拟器在CPU核心上的运行时间比实际执行的时间快几个数量级，将一个模型的搜索时间减少到一个小时以内。该模拟器使SwapAdvisor的GA能够直接优化端到端的执行时间。

4　SwapAdvisor设计

概述。图2给出了SwapAdvi- sor的架构，它与现有的DNN框架（在我们的实现中是MXNet）集成。给定一个数据流图，SwapAdvisor根据该图挑选任何合法的时间表和内存分配作为初始值，并将它们传递给交换规划器，以确定什么张量需要交换和何时交换。交换规划器的结果是一个增强的数据流图，其中包括额外的换入和换出操作符以及额外的控制流边。额外的边是为了确保最终的执行顺序符合给定的时间表和计划器的交换时间。

为了优化，增强后的图被传递给Swa-pAdvisor的数据流模拟器来估计整体执行时间。SwapAdvisor的基于GA的搜索衡量的是

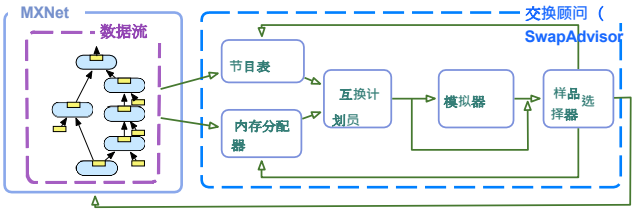


图2. SwapAdvisor的系统概述

许多内存分配/计划组合的性能，并为交换计划器提出新的分配/计划候选。一旦交换计划被充分优化，最终增强的数据流图将被交给框架进行实际执行。

本节描述了交换计划器的输入（第4.1节），并解释了计划器如何在特定的时间表和内存分配下实现性能最大化（第4.2节）。后面一节（第5节）讨论了基于GA的优化。

4.1　操作员时间表和内存分配

除了数据流图之外，交换计划器还需要输入操作者时间表和内存分配。

操作符时间表。当使用单个GPU时，该框架可以根据时间表向GPU发出操作符，以保持GPU的忙碌。事实上，像MXNet这样的框架通常执行拓扑排序来安排运算器。

NVIDIA最近的GPU支持多个 "流"。Swa- pAdvisor使用3个流：一个用于执行GPU执行，一个用于向CPU交换张量，还有一个用于从CPU交换张量。由于GPU-CPU之间的通信是双工的，以这种方式使用时，所有三个流可以同时进行。相比之下，如果要使用多个流进行计算，如果没有足够的GPU计算资源用于并行执行，这些流就不能同时执行。我们观察到，在我们测试的所有DNN模型中，使用一个以上的流进行计算没有任何性能优势。这一观察结果也得到了其他人的认同[40]。

内存分配。我们需要配置内存池并指定给定数据流图的内存分配。内存池由许多不同大小的类组成，每个类被分配一定数量的固定大小的张量对象。给定一个数据流图G，其中包含每个操作者需要的所有输入/输出张量的大小，内存分配方案可以通过指定两件事来定义。1）从G中的每个张量尺寸到内存池支持的某个尺寸类别的映射。2）支持的大小类的集合，以及分配给每个类的张量对象的数量。作为一个例子，图1(b)(c)中的粗粒度分配方案只有一个大小类（2MB），有5个对象，并将每个1MB或

2MB张量映射到2MB大小的类中。图1(d)(e)中的细粒度方案有两个大小类（1MB和2MB），分别有8个和1个对象，并将每个1MB张量映射到1MB大小类中，每个2MB张量映射到2MB大小类中。

4.2 互换计划

交换计划器被赋予数据流图以及有效的操作者时间表和内存分配方案。它的工作是在给定的时间表/分配组合下找到具有最佳性能的交换计划。特别是swap规划器决定。1）在内存压力下，哪些内存驻留的张量需要换出，2）何时执行换入或换出。

哪些张量要被换掉？在高水平上，交换计划员使用Belady的策略来挑选在未来最长时间内不需要的张量来交换。由于计划者得到了时间表，所以可以看到未来的情况。Belady的策略对于缓存替换来说是最理想的，而且在我们的环境中也很有效，因为它给了计划者足够的时间在下次使用前把张量换回来。具体来说，计划者根据时间表中的顺序扫描每个操作者，并跟踪输入/输出张量对象的集合，这些对象由于执行操作者的序列而成为内存中的常驻对象。在添加大小为s的张量时遇到内存压力（即在s的大小类中没有空闲对象），计划者从与s相同的大小类中选择张量进行交换。如果有多个候选对象，计划员会选择一个在最远的将来会被使用的对象。

在我们使用贝拉迪的策略时，有一个注意点。设置。假设张量T_i --最后一次被运算符op_i使用--被选择来为张量T_j 腾出空间，这是当前运算符op_j 的输入张量。因此，T_i 最早可以被换出的时间是*运算符*$_i$ 结束的时候。如果运算符op_i 和op_j 在时间表中的时间太近，那么在运算符op_j 需要T_j 之前，几乎没有时间将其换入，因为其内存空间要在T_i 被换出之后才可用。作为一种补救措施，当选择一个候选张量来交换时，计划者从那些最近被使用过的张量中挑选出至少一个阈值的时间。

DNN训练是迭代式的，但交换计划者只得到一次迭代的数据流图。在每次迭代结束时，除参数张量外的所有张量都可以被丢弃。然而，为了确保同一个交换计划可以在多次迭代中使用，我们必须确保迭代结束时GPU内存中的参数张量集与开始时相同。为了实现这一点，我们进行了一次双通道，即扫描计划表以规划两次交换。在第一遍中，我们假设GPU内存中没有参数张量，必须在其第一次使用前被交换进来。在第一遍结束时，参数张量的一个子集成为内存中的常驻参数，我们将其称为初始常驻参数。然后我们进行

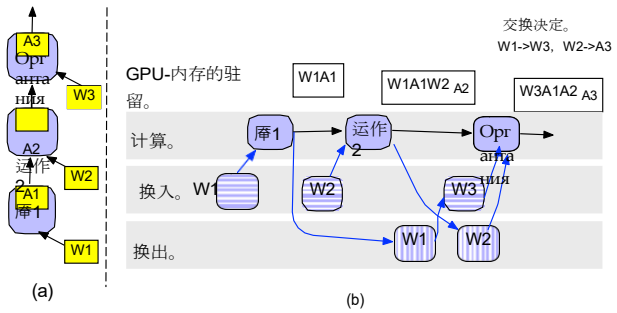


图3.一个简单数据流图的交换计划实例。所有的张量都有单位大小，GPU的总内存是4个单位。

第二遍，假设初始居民参数在计划开始时存在于内存中。在第二遍中，如果有额外的内存压力，而在第一遍中没有发生，我们就从初始驻留参数集合中删除一个参数张量来解决压力。最终交换计划的初始GPU驻留参数不包括在第二遍中删除的参数。

什么时候换入和换出？我们之前讨论的搜索策略已经确定了成对的张量，以便在必要时换出和换入，从而根据时间表执行每个运算符。为了使计算和通信重叠最大化，我们希望尽可能早地完成一对换出和换入，以便不阻碍时间表中相应算子的执行。然而，我们还必须确保换入/换出的时间是安全的。

我们用一个3节点数据流图（图3(a)）和时间表op_1，op_2，op_3 （图3(b)）来说明计划器如何控制交换时间。为简单起见，我们假设例子中的所有张量都是1个单元的大小，GPU的总内存大小为4个单元。为了执行*操作*$_1$，我们必须换入参数张量W_1，因此计划者为换入W_1 增加了一个新的数据流节点，该节点将在专门用于换入的GPU流中运行。同样地，也为W_2 增加了一个换入节点。我们注意到，有足够的内存来容纳*操作*$_1$ 和*操作*$_2$ 的输入/输出张量。然而，为了运行*操作*$_3$，我们需要空间来交换W_3，并为A3分配空间。计划员选择W_1，以使为W_3 （简称$W_1 \rightarrow W_3$）腾出空间，选择W_2 为A_3 （简称$W_2 \rightarrow A_3$）腾出空间。让我们首先考虑$W_1 \rightarrow W_3$ 的情况。计划者增加了两个数据流节点W_1 （换出）和W_3 （换入）。从W_3 (swap-in)到op_3 的控制流边被添加，以确保只有在W_3 进入GPU内存后才开始执行操作。从W_1 (swap-out)到W_3 (swap-in)的边被添加，以确保只有在相应的swap-out完成后内存变得可用时才开始swap-in。此外，还包括一条从*操作*$_1$ 到W_1 （换出）的边，因为在*操作*$_1$ 完成使用之前，W_1 不能从内存中移除。

$W_2 \rightarrow A_3$ 的情况类似，只是计划者不需要为A_3 添加交换节点，因为它是由操作者创建的。由此产生的增强数据流图可以传递给框架的数据流引擎来执行。

5　通过遗传算法进行优化

5.1　算法概述

遗传算法（GA）旨在通过交叉、变异和选择等受自然界启发的机制来进化和改进一个个体的轮胎群[5, 8, 18]。在SwapAdvisor中，个体的染色体由两部分组成：一个操作程序和一个内存分配。第一代个体是随机产生的，种群的大小由一个超参数决定，即N_p。为了创造新的一代个体，我们对当前一代的染色体进行交叉和变异。交叉采用一对亲代染色体，通过结合亲代的特征产生新的个体，这样子女就能从亲代那里继承 "好 "的特征（概率上）。在SwapAdvisor中，每次交叉产生两个新的时间表和两个内存分配，从而产生4个孩子。然后我们对子代进行变异，这对GA摆脱局部最小值和避免过早收敛至关重要[5, 8, 18]。产生的变异子代被交给交换规划器，以生成带有交换节点的增强数据流图。我们使用一个定制的数据流模拟器来执行增强的图并获得执行时间，这被用来衡量个体的质量。最后，GA在这些个体中选择N个$_p$。

目前的人口能够生存到下一代。

选择方法。如何选择生存的个体在GA中是至关重要的。如果我们只选择最好的个体来生存，那么种群就会失去多样性并过早地收敛。

SwapAdvisor的选择考虑到了个体的质量，以确定其生存的概率。假设一个个体的执行时间为t，我们将其正常化的执行时间定义为$t_{norm} = (T_{Best} - t)/T_{Best}$，其中$T_{Best}$是迄今为止所有个体中的最佳时间。一个个体的生存概率由$sof\ tmax$函数决定。

$$\frac{\bar{A}\bar{A}\bar{A}}{e \quad _j}$$

$$概率_i = \frac{}{\sum_{j=1}^{S} tnorm} \text{ for } i = 1..S \qquad (1)$$

在公式中，S是选择前的种群大小（通常大于N_p）。我们使用基于softmax的选择，因为我们的实验表明，与流行的锦标赛选择相比，它能达到更稳定的结果[5, 8, 18]。

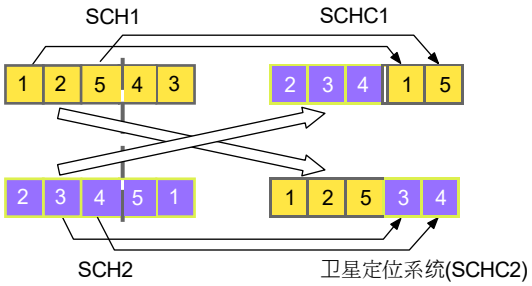


图4.跨越SCH_1和SCH_2。数据流图中有五个节点。

5.2　创建新的时间表

编码。由于日程表是数据流图（G）的拓扑排序，自然可以将日程表编码为一个列表SCH，其中SCH中的每个元素都是G中的一个节点ID。

交叉。我们借用[43]的想法，通过交叉两个父表，即SCH_1和SCH_2，来创建两个子表。我们通过一个例子来解释，如图4所示。首先，随机选择一个交叉点，CR。在这个例子中，CR=3。为了创建子SCH_{C1}，交叉点从SCH_2（SCH_2 [1 ... CR] = [2, 3, 4]）中抽取一个片断，作为SCH_{C1}的第一个部分。不在SCH_{C1}中的节点根据其在SCH_1中的顺序被填入。在本例中，节点1和5不在SCH_{C1} = [2, 3, 4]中，因此我们将它们填入SCH_1的剩余槽中，按这个顺序。SCH_{C2}可以通过同样的方法来创建，但与SCH_1和SCH_2的部分不同，如图4的底部所示。该算法保证SCH_{C1}和SCH_{C2}是G的拓扑排序[43]。

变异。突变时间表的一个简单方法是随机改变一个节点在列表中的位置，只要结果仍然是拓扑排序[43]。然而，我们已经观察到，如果我们在一次突变中对多个节点进行突变，GA的效果会更好（例如，对RNNs来说，性能提高2倍以上）。

SwapAdvisor的突变算法模仿了数据流调度器。它维护着一个准备好的集合，其中包含了所有准备好的节点（所有的前任节点都被排除了）。突变算法的核心功能是根据以下两个条件从就绪集中选择一个节点。首先，在概率为P的情况下，发生突变。在这种情况下，算法从准备集中随机选择一个节点。否则，该算法将从

在原计划中最早执行的准备集

(未变异)。一个被选中的节点被视为 "已执行"。当所有的节点都被 "执行 "时，该算法终止。变异算法产生了一个新的时间表，该时间表主要遵循输入的时间表，但其中一些节点的安排有所不同。

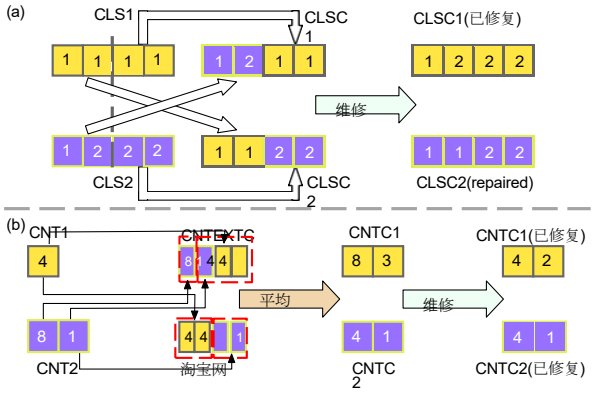


图5.跨越（CLS_1，CNT_1）和（CLS_2，CNT_2）。有12MB内存，有四种不同大小的张量，1MB、2MB、3MB和4MB。

5.3 创建新的内存分配

编码。内存分配控制着如何将每个尺寸映射到一个尺寸类，以及为每个尺寸类分配多少个对象。尽管使用散列映射将张量大小映射到大小类是很自然的，但这样做会丢失不同张量大小之间的相对大小信息，从而更难进行有效交叉。我们使用两个列表，*CLS*和*CNT*，来表示张量尺寸-类的映射。

让*TS*是在数据流图中观察到的唯一张量尺寸的排序列表。*CLS*是一个与*TS*相同长度的列表，*CLS*中的i_{th}项（*CLS*[*i*]）是一个正整数，代表尺寸为*TS*[*i*]的张量的尺寸类。因此，大小类的数量是*Max*(*CLS*)。*CNT*是一个列表，代表为每个大小类分配的张量对象的数量。因此，*CNT*的长度是*Max*(*CLS*)。作为一个例子，图1的数据流图只有两种不同的张量大小，因此*TS* = [1MB, 2MB]。有5个2MB对象的粗粒度分配对应于*CLS* = [0, 0]，表明1MB和2MB的大小都被映射到同一个大小类，ID为1，对象大小为2MB。*CNT* = [5]包含分配给*id* = 0...的每个大小类的对象数量。*Max*(*CLS*)。

潜在的*CLS*列表的数量是$O(N^N)$，其中*N*是独特的张量的数量[52]。这样一个巨大的搜索空间会严重削弱GA的性能。我们修剪了
通过施加限制，*CLS*必须是一个单调递增的序列，并且每个*CLS*[*i*]都等于或大于*CLS*[*i* − 1]，从而实现了搜索空间。其直觉是，当连续的尺寸是
映射到相同或相邻的大小类。这一限制将搜索空间从$O(N^N)$削减到$O(2N)$。

交叉。我们首先用一个例子解释如何交叉使用两个*CLS*，如图5所示。交叉前有两个*CLS*（CLS_1和CLS_2）和四个不同的张量大小。1MB、2MB、3MB和4MB。CLS_1 = [1, 1, 1, 1]意味着所有四个张量大小都属于同一个大小类，即4MB。和CNT_1 = [4]表示有四个4MB的张量对象被分配。CLS_2 = [1, 2, 2, 2]表示有两个大小类，1MB和4MB。有8个1MB张量对象和1个4MB张量对象，因为CNT_2是[8, 1]。

交叉随机选择一个交叉点，*CR*对准...
父级列表CLS_1和CLS_2。第一个子尺寸类映射CLS_{C1}是由CLS_2[1 . . . *CR*]和CLS_1[*CR* + 1 . . *N*]连接而成。第二个子类映射CLS_{C2}是由CLS_1[1 . . .*CR*]和CLS_2[*CR* + 1 . . .*N*]连接而成。图5(a)显示了*CLS*的交叉情况。
在图中，*CR*是2，结果，是CLS_{C1}（[1, 2, 1, 1]）。和CLS_{C2}（[1, 1, 2, 2]）。请注意，CLS_{C1}并非单调增长。因此，我们对其进行修复，以确保新的大小类
映射仍然是有效的。我们的修复方法是将有问题的*CLS*中的元素增加到最小的数量，从而使产生的序列变得有效。在图5(a)中，我们将CLS_{C1}的第3和第4个元素增加1，这样，修复后的CLS_{C1}变成[1, 2, 2, 2]。

同样的杂交方案不能直接用于
*CNT*的长度取决于相关的*CLS*的内容。因此，我们将一个*CNT*扩展为一个扩展的CNT_{EXT}，它的长度与*CLS*（和*TS*）相同。*CNT*捕捉到每个大小类分配了多少张量对象，而CNT_{EXT}则表明每个张量大小可以使用多少张量对象。例如，在图5中，CLS_2是[1, 2, 2, 2]，这意味着1MB属于1MB大小的类，2MB、3MB和4MB属于4MB大小的类。CNT_2是[8, 1]，$CNT_{EXT\,2}$就可以看成是[8, 1, 1, 1]。$CNT_{EXT\,1}$是[4, 4, 4, 4]。我们应用同样的技术来越过前述的*CNT*。图5(b)显示了孩子1和孩子2的扩展CNTs的结果。例如，$CNT_{EXT\,C\,1}$是[8, 1, 4, 4]，最后三个元素属于同一个大小类（根据CLS_{C1}）。我们对同一大小类的所有元素进行平均，得到该大小类的计数。因此，得出的CNT_{C1}是[8, 3]。

与*CLS*类似，一个新的*CNT*可能需要被修复。对于
例如，CNT_{C1}是无效的，因为内存消耗是20MB（8 ∗ 1 + 3 ∗ 4），超过12MB的内存容量。我们修复*CNT*的方法是将每个元素反比例地减少到该元素对应的大小等级。例如，原始的CNT_{C1}是[8, 3]，修复后的CNT_{C1}是[4, 2]，因为第一个大小类是1MB，第二个大小类是4MB。

变异。与调度突变类似，我们在*CLS*（和*CNT*）中突变一个以上的元素。一个元素是
突变的概率为P

$_{th}$如果*CLS*[*i*]等于*CLS*[*i* − 1]，我们可以将*CLS*[i]增加1，因为减少*CLS*[*i*]会破坏单调增加的特性。如果*CLS*[*i*]等于*CLS*[*i* − 1]+1，我们将*CLS*[*i*]减少1。请注意，i_{th}后的所有元素也需要增加或减少，以保持单调递增的特征。

为了改变*CNT*中的元素，我们使用高斯分布，以原始值为平均值。在高斯分布下，变异后的值在大多数时候都接近于原始值，但在小概率情况下会有很大的变化。变异的*CNT*和*CLS*可能会超过内存的限制。我们使用与交叉相同的方法来修复它们。

6 评价

在本节中，我们评估了SwapAdvisor的性能。以下是我们结果的要点。

- 在训练各种大型DNN时，SwapAdvisor可以达到理想基线的53%-99%的训练吞吐量，而GPU内存是无限的。SwapAd- visor对RNNs来说比在线交换基线的性能高出80倍，对CNN来说高出2.5倍。
- 当被用于模型推理时，SwapAdvisor与分担GPU内存的基线相比，服务延迟降低了4倍。
- 与仅搜索内存分配或仅搜索调度相比，SwapAdvisor的联合优化将交换的训练产量从20%提高到1100%。

6.1 实验设置

原型实现。SwapAdvisor是基于MXNet 1.2的。GA和模拟器是用Python写的（4.5K LoC）。我们实现了一个并行的GA--每个进程在一个CPU核上对一部分样本进行交叉、变异和模拟。

对于MXNet，我们修改了调度器，以确保换入和换出操作是在计算流之外的两个独立的GPU流上运行。我们还实施了一个新的内存分配器，它遵循SwapAdvisor的结果。对MXNet的修改是1.5K LoC。

测试平台。我们在一个拥有72个虚拟CPU核心和144GB内存的EC2 c5d.18xlarge实例上运行SwapAdvisor。SwapAdvisor的结果是在一个EC2 p3.2xlarge GPU实例上执行的，该实例有一个NVIDIA V100 GPU，有16GB GPU内存和8个虚拟CPU核，有61GB CPU内存。CPU和GPU之间的PCIe带宽为单向12GB/s，双向20GB/s。对于内存消耗超过61GB的体验，我们使用一个P3.8xlarge实例，它有244GB的CPU内存，但只利用一个GPU。

遗传算法参数。遗传算法的所有参数都是根据经验用网格搜索法确定的。样本大小设置为144，以便将搜索任务均匀地分配给72个CPU核。变异概率的有效性随不同的模型而变化。然而，在我们的评估中，10%是一个很好的起始搜索点。我们设定GA的搜索时间为30分钟。

评估的DNN模型。ResNet[16]是最流行的CNN之一。一个ResNet包含几个残差块；一个残差块有几个卷积运算符。残余块的激活与前一个块的激活相结合，形成最终的输出。我们使用ResNet-152，一个152层的ResNet，来进行推理实验。Wide ResNet[58]是ResNet的一个加宽版本。卷积算子的通道大小被乘以一个宽尺度。由于内存消耗较大，原始工作将宽泛的ResNet应用于小型图像（32x32）数据集CIFAR-10，而不是ResNet使用的ImageNet（224x224）。在我们的训练实验中，输入图像的大小与ImageNet相同。我们把一个宽的ResNet模型称为WResNet-152-*X*，这是一个具有*X*宽尺度的152层宽的ResNet。

Inception-V4[47]是另一个流行的CNN模型。一个Inception-V4模型包含几种类型的起始单元；一个起始单元有许多卷积分支，所有分支的激活张量被串联起来形成最终输出。我们通过增加一个类似于WResNet的宽尺度参数来扩大该模型。我们用Inception-X这个词来表示具有宽尺度*X*的Inception-V4模型。

与人工设计的ResNet和Inception-V4不同，Nas- Net[60]是由深度强化学习搜索制作的。因此，NasNet的模型结构更加不规则。一个Nas- Net模型由一连串的Reduction和Normal单元组成，在consec- utive单元之间有剩余的连接（与ResNet相同）。一个还原或正常单元就像一个初始单元，但有不同的分支结构。正常单元重复3R次。在最初的设计中，最大的*R*是7。在我们的实验中，我们训练NasNet-25，一个*R*=25的NasNet。

RNN[17]是一种用于训练序列输入（如文本）的DNN。RNN的一个层由一个LSTM单元组成，其中一个单元的输入是序列中的相应元素（例如，字符）。双向RNN（BRNN）[41]是RNN的一个变种。BRNN的一个隐藏层包含两个子层。第一个子层的输入是原始序列，第二个子层的输入是反转的序列。两层的激活张量被串联起来，形成最终的激活。每个子层都有自己的参数。我们用RNN-L-X K（BRNN-L-X K）的符号来表示有*L*层的RNN，一层的参数大小为*X*K。

基线。我们将SwapAdvisor与两条基线进行比较。第一条是理想基线，表示为*理想*。对于理想基线，我们假设GPU内存是无限的。我们通过直接重复使用GPU内存来实现理想基线（从而影响了计算的正确性）。理想基线给出了交换系统可实现的严格的性能上限。

第二条基线是一个在线按需交换系统，称为*ODSwap*，它包含了启发式方法，包括基于LRU的交换和预取，被用于

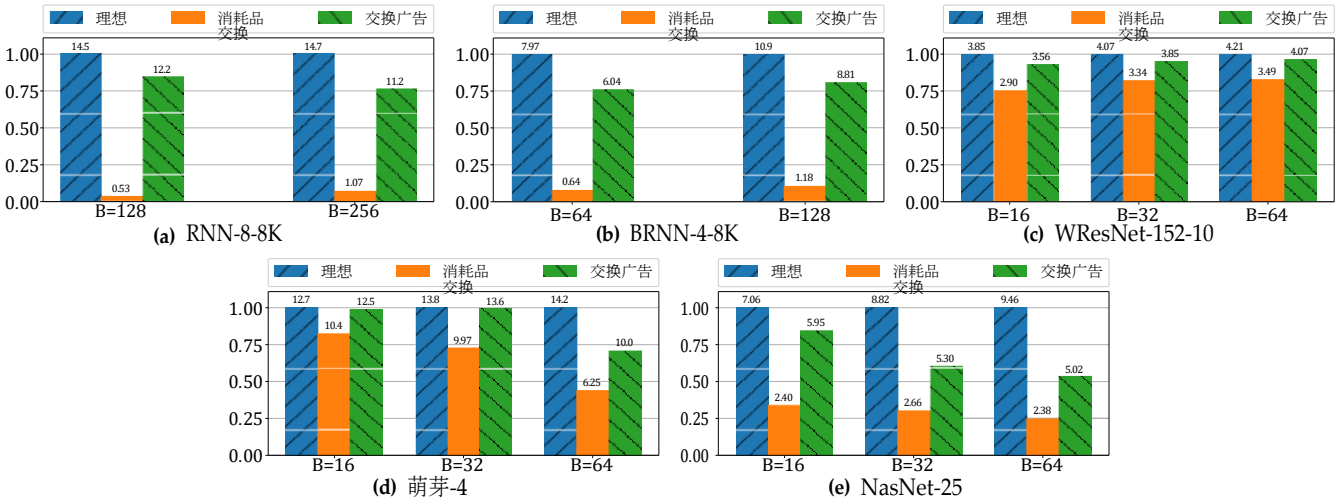


图6.相对于理想性能的归一化吞吐量。每一组条形图代表一个批次的大小。每个条形图上的数字显示的是绝对吞吐量，单位是样品/秒。X轴显示了不同的批次大小。

vDNN[40]和SuperNeuron[49]。具体来说，在第一次训练迭代中，当GPU内存不足以运行基于LRU的下一个节点时，ODSwap会换掉一个张量。在随后的迭代中，ODSwap根据第一次迭代中看到的调度决定进行预取以实现通信和计算的最佳重叠。我们为*ODSwap*重新移植的性能数据忽略了第一次迭代。

6.2　更广泛和更深入的DNN模型训练

表1显示了本节所评估的模型的统计数据。每一行都显示了内存使用量、运算器的数量和不同张量的数量。一个模型的批量大小是图6中最大的那个。

RNN的性能。图6a和6b显示了RNN-8-8K和BRNN-4-8K的吞吐量。SwapAdvisor为RNN和BRNN实现了理想性能的70-80%，而ODSwap的吞吐量只有理想性能的1%以下。对于RNN和BRNN，参数张量是由同一层的LSTM单元共享的。因此，一个执行不同层的LSTM单元的时间表会导致可怕的交换性能，因为系统必须为不同的大参数准备内存。不幸的是，随机生成一个拓扑排序几乎总是导致这样的时间表，这就是MXNet的默认时间表。因此，ODSwap的性能很差。SwapAdvisor能够通过GA找到一个有利于交换的时间表。

CNN的性能。图6c、6d和6e显示了WResNet-152-10、Inception-4和NasNet-1的吞吐量。25.表1显示，WResNet-152-10使用了惊人的180GB内存，但SwapAdvisor和ODSwap都表现良好；SwapAdvisor达到了理想性能的95%，ODSwap达到了理想性能的80%。WResNet-152-10只有26种不同的张量大小，这使得它在做

模型	内存使用量	候选人名单	TensorSizes
WResNet-152-10	180GB	882	26
萌芽-4	71GB	830	64
NasNet-25	193GB	5533	65
RNN-8-8K	118GB	8594	7
BRNN-4-8K	99GB	9034	9

表1.DNN模型的统计数据。CNN模型的批量大小为64，RNN为256，BRNN为128。

内存分配。更重要的是，与RNN/BRNN不同，WResNet的数据流图的拓扑结构更像一条线--只有一个剩余块的跳转链接。因此，调度的选择可能对最终结果影响不大。

另一方面，Inception-4和NasNet-25有超过60种不同大小的张量，使得内存管理更加困难。Inception-V4和NasNet的数据流图的拓扑结构也更加复杂，这一点在第6.1节中有详细说明。与ODSwap相比，SwapAdvisor实现了20%-150%的性能提升。

请注意，对于Inception-4和NasNet-25，当批处理量为16时，SwapAdvisor可以达到理想基线的80%的性能。然而，当批处理量为64时，SwapAdvisor不能达到理想性能的65%以上。当批处理规模为64时，Inception-4和NasNet-25都有许多大的激活张量（>500MB）。再加上大量的张量（>60），SwapAdvisor很难搜索到一个好的池配置，以尽量减少交换开销。NasNet-25的图中也有超过9000个节点，使得它很难调度。因此，在批量大小为64的情况下，SwapAdvisor只达到了NasNet-25理想基线的53%。

与TFLMS的比较。我们还将SwapAdvi- sor和ODSwap与TFLMS[26]进行比较，TFLMS是TensorFlow的交换扩展。不幸的是，TFLMS不能支持图6中的模型。我们评估了WResNet-152-4，因为它是最大的

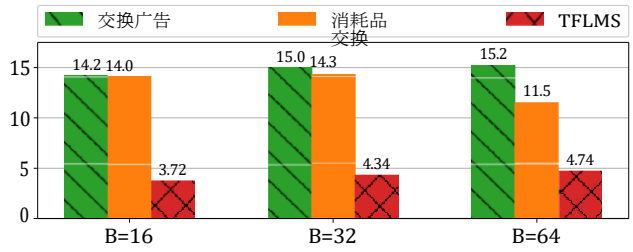


图7.WResNet-152-4的吞吐量比较

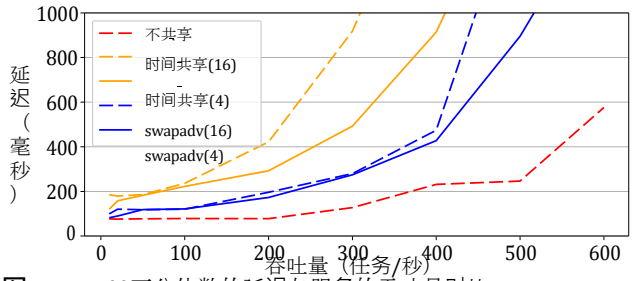


图8.　99百分位数的延迟与服务的吞吐量对比多个ResNet-152模型。

TFLMS的可执行的一个。图7显示，对于所有三种批量大小，SwapAdvisor比TFLMS至少好3倍，而ODSwap比TFLMS至少好2倍。TFLMS的表现很差，因为没有交换参数张量（在WResNet-152-4中是很大的）。这种设计减少了GPU存储激活张量的内存容量，导致激活张量的交换更多。

6.3　DNN模型推理评估

MemSize/Batch	1	16	64
64MB	**0.024s**	不适用	不适用
128MB	**0.022s**	**0.077s**	不适用
192MB	**0.018s**	**0.044s**	不适用
512MB	*0.017s*	***0.040s***	**0.238s**
640MB	*0.017s*	***0.040s***	***0.123s***

表2.不同批次大小和GPU内存大小的ResNet-152推理时间。

模型推理（服务）使用的GPU内存比训练模型少得多，因为没有反向传播。然而，SwapAdvisor在几种情况下仍可用于服务。

表2显示了SwapAdvisor如何在不同的批次规模下减少ResNet-152的内存需求。在表中，每个单元格是一个推理迭代的运行时间，对应的可用GPU内存大小。"N/A "表示SwapAdvisor无法在如此小的内存容量下运行推理工作。粗体字的运行时间意味着需要交换来运行作业（内存不足）。带斜体字的运行时间意味着运行时间与使用全部16GB内存容量的性能接近（性能差异<1%）。

一个有趣的实验是当批处理量为1时。批处理量1很少用于集群上的训练或推理，但用于移动设备上的推理却并不罕见。表2显示SwapAdvisor可以将批量大小为1的内存使用量减少到64MB，运行时间开销为40%，或者减少到192MB，开销仅为6%。SwapAdvisor可以帮助将DNN模型装入资源有限的GPU。虽然移动设备上的GPU和CPU之间的通信速度比EC2 GPU实例上的要慢，但V100 GPU也比移动GPU快得多。因此，移动设备上的实际交换开销可能与表2所示的不同。尽管如此，这个结果仍然为SwapAdvisor提供了一个潜在的使用案例。

SwapAdvisor的另一个可能的服务用例是在不同的DNN推断中分担GPU资源。在设定中，一个GPU机器在各个模型之间分担计算和内存。如果我们只对GPU的计算进行时间共享，而对GPU的内存和数据进行分割，那么会怎么样呢？

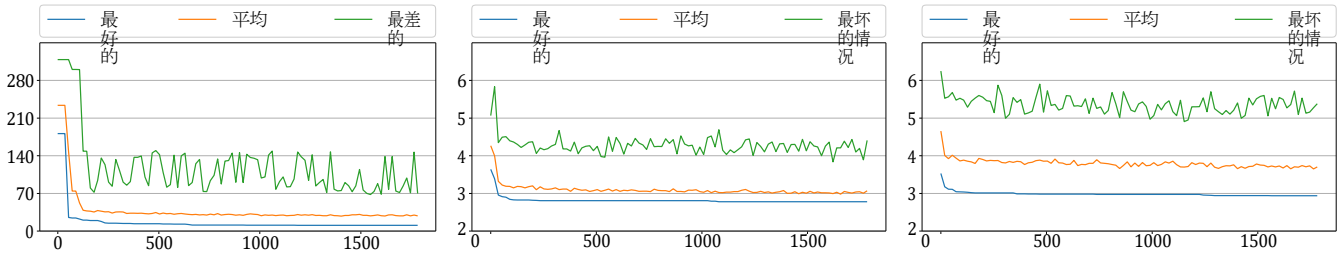
将GPU内存分配给各模型？因为对于一个DNN模型来说，分割的GPU内存容量可能太小。

我们应用SwapAdvisor，以便所有的模型都能在分区的内存上适应。

我们将分时的GPU内存视为基线，并比较客户端的延迟。在评估中，GPU上有多个ResNet-152，每个都有自己的训练参数。我们假设客户到达率遵循泊松分布，并将成批的客户随机分配给不同的模型。图8显示了99百分位数的延迟与GPU机器的吞吐量。图例后面的数字表示有多少个模型在GPU上运行。该图还显示了在GPU上只提供一个模型的延迟，表示为 "not_shared"。

我们可以看到，当4个和16个模型的吞吐量小于400时，SwapAdvisor的99百分位延迟最多比 "not_shared "慢2倍。另一方面，当吞吐量为300时，"time_shared "的延迟比16个模型的 "not_shared "慢了8倍。当吞吐量大于400时，在一个GPU上服务几个ResNet-152可能不是一个明智的决定，因为SwapAdvisor和 "time_shared "的延迟都急剧增加。

SwapAdvisor的主要好处是，它将内存拷贝与计算重叠。另一方面，基线必须在当前模型执行后为下一个模型交换参数张量。如果系统能够预测下一个模型的执行，"时间共享 "就有可能为下一个模型预取参数。有了这样的任务调度器，基线可能会比SwapAdvisor更出色。然而，当任务调度器预测错误时，SwapAdvisor仍然可以用来减轻潜在的开销。



(a) RNN-8-8K　**(b)** 萌芽-4　**(c)** Inception-4 (w/o restricted size-class mapping)

图9.搜索结果与搜索时间的关系。X轴是搜索时间，Y轴是运行时间（秒/迭代）。

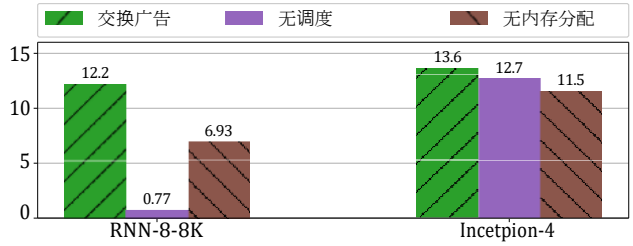


图10.在RNN和Inception-V4的不同搜索设置下，SwapAdvisor的吞吐量。

6.4 SwapAdvisor的设计效果选择

调度和内存分配的有效性。我们希望看到优化调度和内存分配的重要性。图9显示，对于一个RNN模型来说，搜索一个有利于交换的时间表是非常重要的。如果不搜索一个好的时间表，SwapAdvisor只能达到完全搜索时的7%的性能。另一方面，图10显示，内存分配对Inception-V4的性能影响比调度更大。在没有搜索调度的情况下，SwapAdvisor仍然可以达到完全搜索的93%的性能。图10表明，优化交换的调度和内存分配是很重要的，因为这两个部分的有效性在不同的模型中是不同的。

遗传算法的性能。图9显示了GA的搜索性能。在图中，有三条线，分别代表此刻所有活着的样本（144个样本）的最佳、平均和最差模拟时间。第一代样本是随机产生的。一般来说，GA可以在100秒内找到一个好的解决方案，因为平均和最佳模拟时间都在前100秒内迅速收敛。然而，最差（或平均）结果和最佳结果之间的差异表明，人口仍然保持着二元性，允许GA在剩余时间内逐步优化样本。

图9b和图9c都显示了对Inception-的搜索。

4，但图9c假设，如果张量大小映射不受限制。我们可以看到，图9c中所有的最佳、平均和最差结果都比图9b差，证明了

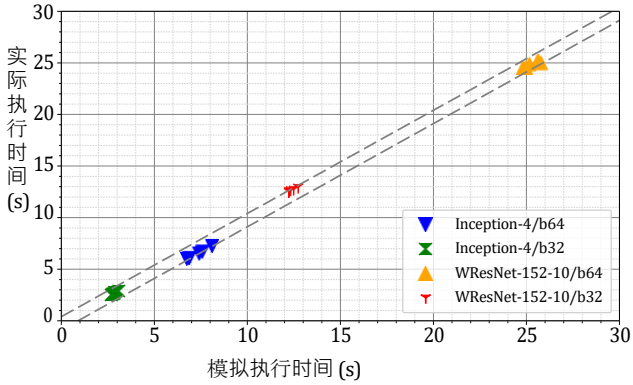


图11.两种不同模型的模拟结果的模拟执行时间与实际执行时间的对比，批量大小为32和64。

限制搜索空间有助于SwapAdvisor找到更好的结果。限制搜索空间的有效性对所有被评估的模型都是通用的。

6.5 模拟器的准确性

图11显示了20个随机选择的仿真结果和它们相应的实际执行时间。有两个模型，Inception-4和WResNet-152-10，有两种不同的批处理量，即32和64。因此，总共有4个不同的实验，每个实验有5个数据点。两条虚线描述了仿真不准确的界限。上面的虚线表示实际执行比模拟慢4%，下面的虚线表示实际执行比模拟快12%。例如，批处理量为32的所有数据点WResNet-152-10（WResNet-152-10/b32）都比较接近上虚线。因此，WResNet-152-10/b32的模拟比实际执行快4%左右。

在图11的20个数据点中，有19个数据点主根据模拟和实际执行，保持相同的相对性能。唯一的例外是最右边的数据点（属于WResNet-152-10/b64）。当两个数据点根据模拟和实际执行有不同的相对性能时，它们的实际执行时间差异在更大的评估中小于10%。

比图11所示的数据点集（在图11中，最大的差异是2%，对应于最右边的两个数据点）。这些结果表明，使用模拟性能是加速基于GA的搜索的一个有效方法。

7 相关工作

DNN的互换。 vDNN[40]交换了所有的激活张量或只交换了所有的卷积张量。TFLMS[26]也只交换激活。[30]使用激活张量及其损失函数节点的关键路径长度作为启发式方法来决定交换哪些激活张量。[57]是TensorFlow的一个按需交换机制。它的启发式方法是将以前的迭代中的张量交换到主机内存中，这个策略只适用于RNNs。

上述工作都没有交换参数，因此不能支持一个大型模型。SuperNeurons[49]采用了一种不同的方法；它结合了交换和重新计算。然而，SuperNeurons把交换限制在对话操作者上。这一决定使SuperNeurons无法支持具有大参数的RNN模型。相比之下，SwapAdvisor可以支持各种更深、更广的DNN模型。

GPU内存限制的替代方法。 存在一些不依靠交换来减少内存消耗的方法。第一个方向包括用低精度浮点数计算[12, 24]、量化和参数压缩[3, 9, 13-15, 20]。[33]观察到CNN推断的激活张量之间的相似性，并提出重用激活张量以加快性能和减少内存消耗。这些技术要么影响模型的准确性，要么需要大量的超参数调整，而交换确实会影响结果。

另一种方法是重新计算。重新计算利用了激活张量可以被重新计算的事实。因此，[2, 11, 29]在前向传播中最后一次使用激活张量后，对其进行去分配，然后在需要时重新计算激活张量。虽然重新计算可以用于较深的模型和大的输入数据，但它不能支持较宽的模型，因为大的参数张量占据了内存，无法重新计算。

最后，用多个GPU训练DNN模型是一项积极的研究。将DNN模型并行化的最流行方式是数据并行化[4, 27, 51]。通过数据并行，每个GPU得到一部分输入数据和模型的全部参数。因此，输入张量和激活张量被切片并分配给GPU，有效地重新减少了每个GPU的内存消耗。虽然很容易使用，但数据并行在每个GPU上重复了全部参数，限制了模型的最大参数大小。有。与数据并行相反，模型并行对激活张量和参数张量都有作用[6, 25]。然而，对一个模型应用模型并行制需要大量的工程工作。[22, 23, 50]提出了自动匹配模型并行性并减少与数据流图分析的通信。

调度以减少内存利用率。 一些工作考虑了任务图调度的吞吐量和其资源消耗之间的权衡[44, 45, 53]。这些工作针对的是缓冲区（内存）非常昂贵的情况，而应用程序有一个吞吐量限制（例如视频帧率）。因此，优化的目标不是最大化吞吐量，而是在吞吐量约束下最小化内存消耗。另一方面，SwapAd- visor的目标是在CPU-内存不受约束和GPU-内存消耗受约束的情况下，使执行时间最小化（吞吐量最大化）。SwapAdvisor的关键设计因素是通信和计算的重叠，这在相关工作中没有考虑。

重叠GPU的通信和计算。 对于与DNN无关的其他类型的计算，现有的工作是重叠通信和计算[1, 59]。他们采用了两种模式。1）将粗粒度的内核划分为细粒度的子内核，因此一些子内核可以开始计算，而其他子内核则等待通信，2）使用队列在处理细粒度任务的生产者/消费者之间通信细粒度的计算结果。内核细分或基于队列的方法能够在单个粗粒度的内核或生产者/消费者对中实现重叠。然而，我们过去的经验表明，在将粗粒度的DNN内核划分为细粒度的子内核时，会有非同小可的性能开销。因此，SwapAdvisor依靠预取来实现许多粗粒度内核的数据流图中的过度映射。

在一个GPU上进行多个DNN推断。 TensorRT[34]利用GPU流来运行多个模型推理，目前。英伟达MPS[35]也支持并发的GPU任务，但这些任务不限于DNN推理。TensorRT和MPS都要求用户对GPU内存进行任务分区。SwapAdvisor可以帮助这两个系统减轻内存压力。Salus[56]旨在支持DNN任务间的细粒度GPU共享。Salus分配了一个共享的内存空间来存储激活张量和所有模型的刮擦空间，因为这些内存消耗可以在迭代中最后一次使用后直接放弃。它假定参数张量在GPU内存中，因此可以借用SwapAdvisor的技术来支持GPU上更多（更大）的模型。

计算机系统的遗传算法。 遗传算法已被用于并行或分布式系统的任务安排[19, 42, 43, 46, 48, 54]。SwapAdvisor从现有的工作中吸收了一些想法，例如，如何交叉安排。然而，SwapAdvisor的设置是不同的。现有的工作是为多核或多机系统设计的，其中一个任务可以被安排在不同的核或机上。另一方面，SwapAdvisor中的所有计算任务都在同一个GPU上执行。因此，对于SwapAdvisor来说，只有执行顺序是重要的，从而导致了不同的交叉和变异。

一些工作使用遗传算法来分配异质内存系统中的数据对象（例如SRAM与DRAM）[7, 38]。SwapAdvisor的内存分配也是在为一个内存池分配内存对象之前决定多少个内存池。

8 讨论、限制和未来工作

动态数据流图。 SwapAdvisor的设计需要一个静态数据流图。为了与PyTorch或在TensorFlow的强制执行模式下，我们可以使用[21]中描述的技术提取数据流图。然而，SwapAdvisor目前不能处理数据流图中的任何控制流原语[57]，而这是支持更广泛的DNN所需要的。

一个潜在的解决方案是采用混合互换的方式。我们可以把动态数据流图看作由三种类型的子图组成。第一种是静态子图，它对应于计算的静态部分，总是被执行。第二个是重复的子图（例如，包含一个 "while "循环），它可以执行一次以上。最后一个是带有条件节点的子图（例如，包含 "if/else"）。SwapAdvisor可以直接应用于静态子图，以及使用同样的技术处理多个训练迭代的重复性子图（4.2节）。对于有条件节点的子图，我们可以使用在线按需交换策略（如ODSwap）。对于混合设计来说，仍有一些开放性的挑战（例如，如何从SwapAdvisor到ODSwap的道具切换），这需要进一步调查。

多GPU支持。 SwapAdvisor目前只针对单GPU进行交换，最好能将其扩展到多GPU上。有两种流行的多GPU训练方法：数据并行和模型并行。通过数据并行，每个GPU使用不同的训练数据子批运行相同的数据流图。在这种情况下，我们可以为一个GPU运行SwapAdvisor，同时考虑到n个GPU的可用GPU-CPU带宽是1/n。不幸的是，SwapAdvisor在模型并行方面没有直接的调整。这是因为，与数据并行不同，模型并行在前向/后向传播期间会产生GPU间的通信。由于SwapAdvisor不考虑GPU之间的通信。所产生的交换策略不太可能是最优的。如何将SwapAdvisor推广到模型并行下的多GPU训练，仍然是一个开放的研究问题。

替代搜索方法。 我们选择GA是因为我们发现它在经验上运作良好，而且比一些替代方法（例如模拟退火）快得多。然而，我们还没有探索某些其他有前途的搜索方法，如强化学习[31, 32]。使用GA也迫使我们简化某些设计选择，例如，我们采用基于内存池的分配，因为在GA的搜索中很难加入动态分配器。使用更好的搜索方法来消除这些限制是否有好处，需要进一步研究。

9 总结

我们提出了SwapAdvisor，使DNN模型在有限的GPU内存中得到训练和服务。SwapAdvisor通过优化调度、内存分配和交换计划三个方面实现了良好的性能。为了同时优化调度和内存分配，SwapAdvisor采用了遗传算法来寻找一个良好的组合。对于一个给定的时间表和内存分配，SwapAdvisor的交换计划能够确定什么和什么时候交换张量，以最大限度地提高重叠计算和通信。

鸣谢

这项工作得到了美国国家科学基金会（CNS-1816717）、纽约大学英伟达人工智能实验室（NVAIL）、AMD研究经费以及AWS云计算研究经费的部分支持。我们感谢匿名审稿人对论文的有益反馈，并感谢王敏杰和李承晨在整个项目中进行的有见地的讨论。

参考文献

[1] B.Bastem, D. Unat, W. Zhang, A. Almgren, and J. Shalf.2017.在GPU上用瓦片进行计算时过度映射数据传输。在*2017年 第46届国际并行处理会议（ICPP）*。171-180.

[2] 陈天琪，徐兵，张志远，和Carlos Guestrin。2016.训练具有亚线性内存成本的深度网。*arXiv预印本 arXiv:1604.06174*（2016）。

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David.2015.BinaryConnect:在传播过程中用二进制权重训练深度神经网络。在*第28届神经信息处理系统国际会议论文集-第二卷（NIPS'15）*中。MIT Press, 3123-3131。

[4] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R. Ganger, Phillip B. Gibbons, and et al. 2014。利用有边界的呆滞性来加速大数据分析。在*2014年USENIX年度技术会议（USENIX ATC'14）的会议记录*中。USENIX 协会，37-48。

[5] 劳伦斯-戴维斯。1991.遗传算法手册》。(1991).

[6] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, and et al. 2012。大规模分布式深度网络。在

第25届神经信息处理系统国际会议论文集-第一卷（NIPS'12）。Curran Associates Inc., 1223- 1231.

[7] Keke Gai, Meikang Qiu, and Hui Zhao.2016.云计算中使用遗传算法的成本意识到的异构存储器的多媒体数据分配.*IEEE Transactions on Cloud Computing* (2016), 1-1.

[8] David E Goldberg and John H Holland.1988.遗传算法和 机器学习。*机器学习*3，2（1988）。

[9] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev.2014.*arXiv preprint arXiv:1412.6115*（2014）。

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.2016.*深度学习*。麻省理工学院出版社。http://www.deeplearningbook.org。

[11] Audrundefinednas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves.2016.通过时间的记忆高效反向传播。在*第30届国际神经信息处理系统会议（NIPS'16）论文集*。Curran Associates Inc., 4132- 4140.

[12] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan.2015.具有有限数字精度的深度学习。在*第32届国际机器学习会议论文集-第37卷（ICML'15）*中。JMLR.org， 1737-1746。

[13] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally。2016.EIE: Efficient Inference Engine on Compressed Deep Neural Network.*SIGARCH Comput.Archit.News* 44, 3 (June 2016), 243-254.

[14] Song Han, Huizi Mao, and William J Dally.2015.深度压缩。用剪枝、训练好的量化 和胡夫曼编码压缩深度神经网络。*arXiv 预印本 arXiv:1510.00149*（2015）。

[15] Song Han, Jeff Pool, John Tran, and William J. Dally.2015.为高效的神经网络同时学习权重和连接。在*第28届神经信息 处理系统国际会议论文集-第一卷（NIPS'15）*。麻省理工学院出版社，1135-1143。

[16] 何开明，张翔宇，任少卿，和孙健。2016.用于图像识别的深度残差学习。In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.770-778.

[17] Sepp Hochreiter 和 Jürgen Schmidhuber.1997.Long Short-Term Memory.*Neural Comput.*9, 8 (Nove. 1997), 1735-1780.

[18] John Henry Holland等人，1992年。*自然和人工的适应性系统：生物学、控制、 和人工智能应用的介绍性分析*。麻省理工学院出版社。

[19] Edwin SH Hou, Nirwan Ansari, and Hong Ren.1994.一个用于多处理器调度的遗传算法.*IEEE Transactions on Parallel and Distributed Systems* 5, 2 (Feb 1994), 113-120.

[20] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio.2016.二值化神经网络。*在第30届国际神经信息处理系统会议论文集 （NIPS'16）*。Curran Associates Inc., 4114-4122。

[21] Eunji Jeong, Sungwoo Cho, Gyeong-In Yu, Joo Seong Jeong, Dong-Jin Shin, and Byung-Gon Chun.2019.JANUS: Fast and Flexible Deep Learning via Symbolic Graph Execution of Imperative Programs.In *16th USENIX Symposium on Networked Systems Design and Implemen- tation（NSDI 19）*.USENIX协会，453-468。

[22] Zhihao Jia, Sina Lin, Charles R Qi, and Alex Aiken.2018.探索并行化卷积神经网络的隐藏维度。*在国际机器学习会议*上。

[23] Zhihao Jia, Matei Zaharia, and Alex Aiken.2018.Beyond data and model parallelism for deep neural networks. *arXiv preprint arXiv:1807.05358*（2018）.

[24] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M. Aamodt, Natalie Enright Jerger, and Andreas Moshovos.2016.Proteus:Exploiting Numerical Precision Variability in Deep Neural Networks.*在2016年国际超级计算会议的论文集中。*

(ICS'16).计算机械协会，第23条。

[25] 亚历克斯-克雷泽夫斯基。2014.卷积 神经网络并行化的一个怪招。*arXiv预印本arXiv:1404.5997*（2014）。

[26] Tung D Le, Haruki Imai, Yasushi Negishi, and Kiyokuni Kawachiya.2018.Tflms: *arXiv preprint arXiv:1807.02037* (2018). 通过图重写在tensorflow中支持大模型。

[27] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su.2014.用参数服务器扩展分布式机器学习。*在第11届USENIX操作系统设计与实现会议（OSDI'14）论文集中*。USENIX协会，583-598。

[28] Kim-Fung Man, Kit Sang Tang, and Sam Kwong.2001.*Genetic algorithms: concepts and designs*.Springer Science & Business Media.

[29] James Martens和Ilya Sutskever。2012.用无悬念的优化训练深度和递归网络。*在神经网络。Tricks of the Trade*.Springer.

[30] 陈萌，孙敏敏，杨俊，邱明辉，和顾阳。2017.在TensorFlow上通过GPU内存优化训练更深的模型。In *Proc. of ML Systems Workshop in NIPS*.

[31] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V Le, and Jeff Dean.2018.设备安置的分层模型。(2018).

[32] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean.2017.带有强化学习的设备放置优化。*在第34届国际会议论文集 on Machine Learning - Volume 70 （ICML'17）*。JMLR.org，2430-2439。

[33] 林宁和沈西鹏。2019.Deep Reuse:通过粗粒度计算重用来简化CNN的飞行推理。*在ACM国际超级计算会议（ICS'19）的论文集中*。Association for Computing Machinery, 438-448.

[34] 英伟达。2018.NVIDIA TensorRT.(2018)。https://developer.nvidia。com/tensorrt

[35] 英伟达公司。2019. CUDA Multi-Process Service. (2019). https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

[36] 英伟达公司。2019.NVIDIA NVLINK高速互连。(2019). https://www.nvidia.com/en-us/data-center/nvlink/

[37] PCI-SIG。2019.PCI Express基础规范修订版5.0.（2019年）。https://pcisig.com/specifications

[38] M.Qiu, Z. Chen, J. Niu, Z. Zong, G. Quan, X. Qin, and L. T. Yang.2015.用遗传算法进行混合内存的数据分配.*IEEE Transactions on Emerging Topics in Computing* 3, 4 (Dec 2015), 544-555.

[39] Colin Reeves和Jonathan E Rowe.2002.*遗传算法：原则和观点：GA理论指南*。Vol. 20.Springer Science & Business Media。

[40] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W. Keckler.2016年，vDNN。虚拟化的深度神经网络，用于可扩展、内存高效的神经网络设计。In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture (MI- CRO'49)*.IEEE出版社，第18条。

[41] M.Schuster 和 K. K. Paliwal。 1997. 双向递归神经网络.*IEEE Transactions on Signal Processing* 45, 11 (Nove 1997), 2673-2681.

[42] 哈米尔-辛格和阿卜杜-优素福。1996.使用遗传算法对异质任务图进行映射和调度.In *5th IEEE Het- erogeneous Computing Workshop (HCW'96)*.

[43] Oliver Sinnen.2007.*并行系统的任务调度（Wiley Series on Parallel and Distributed Computing）*.Wiley-Interscience，美国。

[44] Sander Stuijk, Marc Geilen, and Twan Basten.2008.Cyclo-Static和同步数据流图的吞吐量-缓冲权衡探索。*IEEE Trans.Comput.*57, 10 (Oct 2008), 1331-1345.

[45] Sander Stuijk, Marc Geilen, Bart Theelen, and Twan Basten.2011. 情景感知数据流。动态应用的建模、分析和实现。In *2011 International Conference on Embedded Computer Systems:架构、建模和仿真*。404-411.

[46] Sung-Ho Woo, Sung-Bong Yang, Shin-Dug Kim, and Tack-Don Han.1997.分布式计算系统中的任务调度与网路算法.In *Proceedings High Performance Computing on the Information Superhighway.HPC Asia'97*.301-305.

[47] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi.2017.Inception-v4、Inception-ResNet和剩余连接对学习的影响。在*第三十一届AAAI人工智能会议（AAAI'17）的论文集中*。AAAI出版社，4278-4284。

[48] Lee Wang, Howard Jay Siegel, Vwani P Roychowdhury, and Anthony A Maciejewski.1997.使用基于遗传算法的方法在异构计算环境中进行任务匹配和调度。*J.Parallel and Distrib.计算*。47, 1 (1997).

[49] Wang Linnan, Jinmian Ye, Yiyang Zhao, Wei Wu, Ang Li, Shuai-wen Leon Song, Zenglin Xu, and Tim Kraska.2018.Superneurons:用于训练深度神经网络的动态GPU内存管理。在*第23届ACM SIGPLAN并行编程的原则和实践研讨会（PPoPP'18）上*。

[50] 王敏杰，黄建钦，和李金阳。2019.使用自动数据流图分区支持非常大的模型。In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys'19)*.Association for Computing Machinery, Article 26.

[51] Jinliang Wei, Wei Dai, Aurick Qiao, Qirong Ho, Henggang Cui, Gregory R. Ganger, Phillip B. Gibbons, Garth A. Gibson, and Eric P. Xing。2015.用于快速数据并行迭代分析的管理性通信和一致性。在*第六届ACM云计算研讨会（SoCC'15）的论文集中*。计算机械协会，381-394。

[52] Eric W. Weisstein.2010.贝尔数。来自MathWorld - A Wolfram网络资源。(2010)。http://mathworld.wolfram.com/BellNumber。

语境

[53] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit.2007.Cyclo-Static数据流图的缓冲容量的高效计算。In *Proceedings of the 44th Annual Design Automation Confer- ence (DAC'07)*.Association for Computing Machinery, 658-663.

[54] Annie S. Wu, Han Yu, Shiyuan Jin, Kuo-Chi Lin, and Guy Schiavone.2004.多处理器调度的增量遗传算法方法.*IEEE Trans.Parallel Distrib.Syst.*15, 9 (Sept. 2004), 824-834.

[55] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, and Mohammad Norouzi.2016.谷歌的神经机器翻译系统。弥合人类和机器翻译之间的差距。在 *arxiv.org:1609.08144*。

[56] 于培峰和Mosharaf Chowdhury。2019.Salus: Fine-Grained GPU Sharing Primitives for Deep Learning Applications. *arXiv preprint arXiv:1902.04610* (2019).

[57] Yuan Yu, Martín Abadi, Paul Barham, Eugene Brevdo, Mike Burrows, Andy Davis, Jeff Dean, Sanjay Ghemawat, Tim Harley, Peter Hawkins, and et al. 2018。大规模机器学习中的动态控制流。在*第十三届EuroSys会议（EuroSys'18）的论文集中*。计算机械协会，第18条。

[58] Sergey Zagoruyko和Nikos Komodakis。2016.广义的剩余网-工程。*arXiv预印本arXiv:1605.07146*（2016）。

[59] Zhen Zheng, Chanyoung Oh, Jidong Zhai, Xipeng Shen, Youngmin Yi, and Wenguang Chen.2019.HiWayLib:为异构管线计算启用高性能通信的软件框架。In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'19)*.计算机协会 chinery.

[60] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le.2018. 为可扩展的图像识别学习可转移的架构。In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*.8697-8710.