高技术通讯 2023 年 第33 卷 第11 期:1160-1171

doi:10.3772/j.issn.1002-0470.2023.11.004

基于自适应静态数据布局策略的深度学习张量程序自动生成框架①

樊 哲②*** 南子渊*** 郝一帆* 杜子东③* 陈云霁***

(*中国科学院计算技术研究所计算机体系结构国家重点实验室 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 如何确定静态数据布局是深度学习张量程序自动生成框架面临的重大挑战。Ansor 作为目前应用最广泛、最具前景的此类框架,其根据预先指定的单一静态数据布局策略,训练性能预测模型,依据该模型搜索最佳性能的张量程序。但其存在单一策略非最优和性能预测模型不准确的问题。为此,本文提出基于自适应静态数据布局(AL)策略的深度学习张量程序自动生成框架 AL-Ansor。AL-Ansor 在搜索过程中自适应地选取多种静态数据布局策略,共同训练性能预测模型,从而搜索得到性能更高的张量程序。本文以 32 核 Intel Xeon CPU 为目标硬件平台,在多个卷积层上进行实验,结果表明,在同样的搜索次数下,相较于基于 3 种指定静态数据布局策略的 Ansor,AL-Ansor 生成的张量程序分别有 13.81%、12.41%和 16.59%的平均性能提升。

关键词 深度学习;张量程序自动生成框架;静态数据布局策略;自适应策略;性能预测模型

0 引 言

深度学习作为一个热门研究方向,其呈现出多样化和轻量化的发展趋势。多样化趋势主要体现在模型和硬件的不断更迭。模型发展如 AlexNet[1]、ResNet[2]、CapsNet[3]和 Transformer[4]等;硬件发展如中央处理器(central processing unit, CPU)、图形处理器(graphics processing unit,GPU)、DianNao 系列[5-6]、张量处理器(tensor processing unit,TPU)[7]以及各种针对特定算法的专用加速器[8-10]。轻量化趋势主要体现在深度学习应用从云端到边端的迁移,即越来越多的深度学习应用被部署到算力受限的边缘端设备[11-13]。这对深度学习应用的算法优化提出了越来越高的要求。上述发展趋势带来了深度学习应用的编程和性能之间的矛盾。

深度学习张量程序(张量算子的低层级实现[14])的自动生成框架致力于解决编程与性能之间的矛盾。其被广泛用于深度学习编译器(如张量虚拟机(tensor virtual machine,TVM)[15])以自动生成高性能的张量程序。其通过搜索的方式找寻在指定硬件后端上性能最优的张量程序,之后交由对应的硬件编译器生成二进制指令,在减轻用户编程负担的同时尽量保证性能。

为了生成高效的深度学习张量程序,必须考虑静态数据的布局问题。原因主要有 3 个方面:(1)对静态数据的布局变换可发生在编译时期,而不占用程序的运行时间;(2)在深度卷积神经网络的前向推理(inference)过程中,静态数据(如权值、偏置等)参与的运算高达总运算量的 90%[16-17];(3)数据布局可以提升程序访存局部性和硬件资源利用

① 国家重点研发计划(2020AAA0103802),国家自然科学基金(61925208,61732020,U19B2019),中国科学院战略性先导科技专项(XDB32050200),北京智源人工智能研究院以及北京市科技新星计划(Z191100001119093),中国科学院稳定支持基础研究领域青年团队计划(YSBR-029)和中国科学院青年创新促进会和科学探索奖资助项目。

② 男,1995 年生,博士生;研究方向:计算机系统结构;E-mail:fanzhe@ict.ac.cn。

③ 通信作者,E-mail:duzidong@ict.ac.cn。

(收稿日期:2022-03-31)

率[18-20]。因此,静态数据的布局对深度学习张量程序的性能影响十分显著;而如何确定合适的静态数据布局,使得程序执行效率最高,是深度学习张量程序自动生成框架面临的重大挑战。

作为目前应用最广泛、最具前景的深度学习张量程序自动生成框架,Ansor[14]应对上述挑战的方案是:根据预先指定的单一数据布局策略,训练性能预测模型,依据该模型搜索最佳性能的张量程序。但其存在单一策略非最优和性能预测模型不准确的问题。具体地,单一的数据布局策略无法使 Ansor 在所有任务上都生成性能最优的程序,而对于一个任务,其最优策略无法预先确定,因此预先指定单一策略的 Ansor 生成的程序仍有性能提升空间(即单一策略非最优的问题)。另外,指定一种策略后,Ansor 会依据性能预测模型 C 从任务的程序空间(即搜索空间)中采样一个程序 A,用对 A 进行布局变换得到的程序 X 去训练 C,以指导下一次采样,直到搜索过程结束。最后 Ansor 输出搜索过程中遇到的性能最优程序 X^*。从优化角度看,X 所在程序空间即为优化空间。C 并非直接在优化空间中寻优,而是通过在搜索空间中采样,间接在优化空间中寻优。优化空间的程序因经过了数据布局,访存局部性要比搜索空间的程序好,因而优化空间和搜索空间在访存特性上存在巨大差异。这种差异导致 Ansor 得到的程序 X^* 仍有性能提升空间(即性能预测模型不准确的问题)。

针对上述问题,本文提出基于自适应静态数据布局(adaptive layout,AL)策略的深度学习张量程序自动生成框架 AL-Ansor。其通过自适应地选取多种静态数据布局策略来解决单一策略非最优的问题;通过多种策略共同训练性能预测模型 C 来解决模型不准确的问题。具体地,AL-Asnor 可先以直接寻优的布局策略预训练 C,将搜索空间限制在静态数据访存局部性较好的程序子空间。然后,再以间接寻优的布局策略(此时搜索空间和优化空间的不一致已被缓解)微调 C,使之更容易在搜索空间中采样出程序 A^*,其经布局变换后的程序 X^* 的性能最优。本文在 32 核 Intel Xeon CPU 上对多个卷积层进行实验验证,结果表明,在同样的搜索次数下,相较于 3 种基于单一策略的 Ansor,AL-Ansor 生成的张量程序分别有 13.81%、12.41% 和 16.59% 的平均性能提升。

综上,本文的主要贡献有以下 3 点:

(1) 分析了 Ansor 的工作原理,发现了其单一策略非最优和性能预测模型不准确的问题。

(2) 提出了自适应静态数据布局策略 AL 以及基于此策略的张量程序自动生成框架 AL-Ansor。

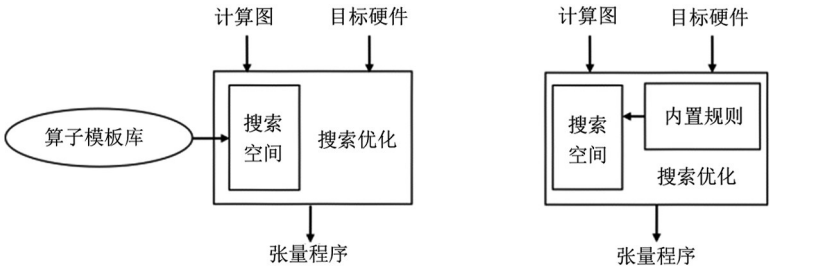
(3) 在典型深度神经网络的多个卷积层上进行实验验证,证明了 AL 策略的有效性。

1　背　景

1.1　张量程序自动生成框架

由于深度学习模型和硬件的多样化发展,基于搜索的张量程序自动生成框架成为一个重要研究方向。这些框架以描述深度学习模型/算子的计算图(computational graph)和目标硬件信息作为主要输入,以优化硬件执行时间为主要目标执行搜索过程,来自动生成对不同硬件后端具有通用表达能力的张量程序。

张量程序自动生成框架按照其搜索空间的定义方式可以分为 2 类:基于模板搜索(如文献[21])和无模板搜索(如文献[14,22]),如图 1 所示。基于模板搜索的搜索空间由用户自定义模板来指定;而无模板搜索的搜索空间无需用户参与指定。AutoTVM 是基于模板搜索的一种典型框架。在该框架下,用户需要针对不同硬件自定义不同算子的搜索模板。搜索模板的内容包括算子的实现方式,输入数据应该如何变换布局,以及输入数据以什么方式(循环结构、循环顺序、向量化、并行化等)进行计算等。算子搜索模板内的不确定因素(循环的拆分因子、是否向量化等)定义了输入计算图对应的张量程序搜索空间。但这样以自定义模板方式定义的空间是非常受限的,也不易扩展[14]。Ansor 是无模板搜索的一种典型框架。其搜索空间由内置的与算子解绑的通用派生规则(derivation rules)定义;这些规则可作用于任意输入计算图来实现对计算图的改写,每个改写的计算图对应一种张量程序实现,这些



(a) 基于模板的张量程序自动生成框架　(b) 无模板的张量程序自动生成框架

图 1　张量程序自动生成框架示意图

构成输入计算图的张量程序空间,即搜索空间。综上,Ansor 无需为每个计算图专门定义搜索空间,也无需用户参与,因此,其更容易应用于新的算子,也缓解了用户的编程负担,是目前使用最广泛、最有前景的深度学习张量程序自动生成框架。

1.2　数据布局优化

数据布局指的是数据在内存中的摆放顺序。改变一个程序所需数据的布局,会影响程序的访存局部性和硬件的计算资源利用率。数据布局优化,即通过改变数据布局,改善程序的访存局部性,使整个程序的性能更好。

图 2 展示了矩阵乘程序的一个数据布局优化过程。图 2(a)展示了一种实现 $(M,K)\times(K,N)$ 矩阵乘的程序 A。假设 2 个输入矩阵 $\boldsymbol{U}$、$\boldsymbol{V}$ 以及输出矩阵 $\boldsymbol{Z}$ 都按行(row-major)存储。对 A 的输入数据 $\boldsymbol{V}$ 进行数据布局优化后,会得到 2 个程序:图 2(b)所示的布局变换程序 L 和图 2(c)所示的布局后程序 P。布局变换程序 L 可根据程序 A 对数据 $\boldsymbol{V}$ 的下标访问规则自动生成(Ansor 中的数据布局变换也是这样实现的),使得布局后的数据 $\boldsymbol{V}'$ 在同样的循环结构,即程序 P 中拥有更好的访存局部性。布局后程序 P 和程序 A 的差别如图 2(c)中加粗部分所示,仅在于数据 $\boldsymbol{V}'$。数据布局优化除了使程序 P 拥有更好的访存局部性外,还可以提升硬件计算资源的利用率。对于一个拥有 NI 个 KI 维向量点积运算单元的硬件平台,程序 P 的最内两重循环(即图 2(c)中虚线框部分)可以直接由这些向量点积运算单元处理,而程序 A 由于对数据 $\boldsymbol{V}$ 的访存不连续,无法直接利用这些运算单元。如果数据 $\boldsymbol{V}$ 是编译时可确定的静态数据,则上述数据布局优化称为静态数据布局优化,此时布局变换程序 L 可在编译过程中执行,不会带来运行时开销。

```
IN: V[K,N]
OUT: V'[NO,KO,NI,KI]
for no in [0, NO):
 for ko in [0, KO):
   for ni in [0, NI):
    n=no*NI+ni
    for ki in [0, KI):
     k=ko*KI+ki
     V'[no,ko,ni,ki]=V[k, n]
```

(b) 布局变换程序 L

数据布局优化 →

```
IN: U[M,K], V[K,N]
OUT: Z[M,N]
for mo in [0, M):
 for no in [0, NO):
  for ko in [0, KO):
   for mi in [0, MI):
    m=mo*MI+mi
    for ni in [0, NI):
     n=no*NI+ni
     for ki in [0, KI):
      k=ko*KI+ki
      Z[m,n]+=U[m,k]*V[k,n]
```

(a) 矩阵乘程序 A

```
IN: U[M,K], V'[NO,KO,NI,KI]
OUT: Z[M,N]
for mo in [0, M):
 for no in [0, NO):
  for ko in [0, KO):
   for mi in [0, MI):
    m=mo*MI+mi
    for ni in [0, NI):
     n=no*NI+ni
     for ki in [0, KI):
      k=ko*KI+ki
      Z[m,n]+=U[m,k]*V'[no,ko,ni,ki]
```

(c) 布局后程序 P

图 2　数据布局优化示例(虚线框部分可进行并行加速,从而提高硬件计算资源利用率;加粗部分展示了 P 和 A 的差异)

对于深度学习应用而言,数据布局优化是提升程序性能的一个重要优化策略。Caffe[23]在实现卷积操作时,先对输入数据进行 im2col 布局变换,再调用高效的通用矩阵乘(general matrix-matrix multiplication,GEMM)接口实现卷积功能。文献[24]为在单指令数据流(single instruction multiple data,SIMD)架构上实现高效的卷积运算,需要将输入布局设置为 NCHWc,权值布局设置为 KCRSck。LDL-NDK(neural network development kit with labled data layout)[25]使用标签标记静态数据布局信息,在编译时进行布局变换,在深度学习加速器上达到数倍性能提升。CDUCA(computation and data unified compile architecture)[26]的数据优化器模块在部署前对数据进行面向硬件平台的数据布局等优化,提升硬件平台的访存和运算效率。NeoCPU[27]针对神经网络在 CPU 上的前向推理过程,在图级别对各算子的神经元数据布局进行规划,再对静态权值数据进行相应的布局变换,达到高性能。

2　问题定义

为了更加清晰地进行后续论述,本文在此定义并引入一些符号,并对张量程序自动生成框架的静态数据布局问题进行定义。

2.1　任务及程序空间

任务 T　由输入数据、输出数据以及输入数据到输出数据的映射规则共同定义。输入数据根据其内容是否为编译时确定又可以分为静态输入数据和动态输入数据。例如,图 2 所示为一个将 $M\times K$ 维输入数据和 $K\times N$ 维输入数据进行矩阵乘得到 $M\times N$ 维输出数据的任务 MatMul。

程序　解决给定任务(实现输入数据到输出数据映射规则)的一个有限可终止的步骤序列(包括访存步骤和计算步骤)。对于图 2 所示程序 A 和 P,虽然其计算步骤是相同的,但是其输入(分别为 $\boldsymbol{V}$ 和 $\boldsymbol{V}'$)和访存步骤不同,因此是 2 个不同的程序。

任务 T 的程序空间$\mathbb{A}_T$　解决给定任务 T 的所有程序构成的空间。图 2 中程序 A 和 $(L+P)$ 均为任务 MatMul 的程序空间$\mathbb{A}_{\text{MatMul}}$中一点。对于一个具体的搜索过程(如 Ansor 的搜索过程),任务 T 的程序空间受限于搜索空间的生成方式,但为了描述的方便,会将该搜索过程所能探索到的程序空间记为$\mathbb{A}_T$。

任务 T 的布局分离程序空间$\mathbb{S}_T$是对空间$\mathbb{A}_T$中每个程序进行数据布局优化所得到的程序构成的空间。显然,该空间是$\mathbb{A}_T$的一个子空间。在图 2 所示的 MatMul 任务中,对程序 $A\in\mathbb{A}_{\text{Matmul}}$进行数据布局优化后得到的组合程序$(L+P)$为$\mathbb{S}_{\text{MatMul}}$空间中一点。

任务 T 的布局后程序空间$\mathbb{P}_T$是对空间$\mathbb{A}_T$中每个程序进行数据布局优化所得到的布局后程序构成的空间。因此,该空间的程序访存局部性都很好,与空间$\mathbb{A}_T$在程序访存特征上存在较大差异。根据定义,$\mathbb{A}_T$和$\mathbb{S}_T$上的点可以在$\mathbb{P}_T$上找到点与之对应,反之,$\mathbb{P}_T$上的点也都可以在$\mathbb{A}_T$和$\mathbb{S}_T$上找到点与之对应。但$\mathbb{P}_T$并不是空间$\mathbb{A}_T$或$\mathbb{S}_T$的子空间,因为$\mathbb{P}_T$中程序解决的任务并不是 T 本身,而是 T 经过布局变换程序后的子任务(该子任务的输入由于数据布局变换,已不同于任务 T 中对应的输入,如图 2 中的 $\boldsymbol{V}$ 和 $\boldsymbol{V}'$)。图 2(c)所示程序 P 即为$\mathbb{P}_{\text{MatMul}}$空间中一点。

图 3 展示了上述术语的关系。对于一个任务 T,从其程序空间$\mathbb{A}_T$中取一程序 A,其通过数据布局优化会产生布局变换程序 L 和布局后程序 P。其中,程序 $(L+P)$ 属于任务 T 的布局分离程序空间$\mathbb{S}_T$,程序 P 属于任务 T 的布局后程序空间$\mathbb{P}_T$。虚线展示了程序 A 到 P 的数据布局优化路径。反过来,$\mathbb{P}_T$中的每一个程序也都可以在$\mathbb{S}_T$和$\mathbb{A}_T$中找到对应的程序。即对于一个任务 T,这 3 个空间中存在一一对应的路径。

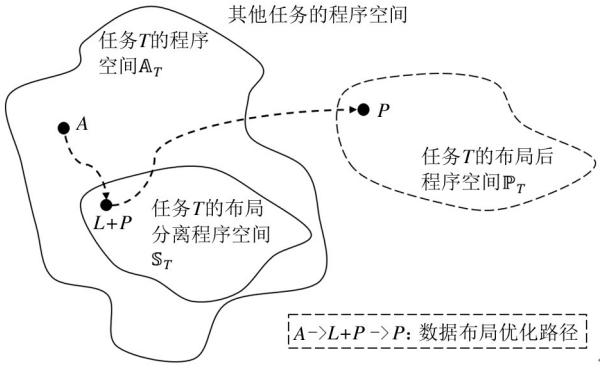


图 3　3 种程序空间的关系以及其中程序的对应关系

2.2　张量程序自动生成框架的静态数据布局

对于张量程序自动生成框架，给定任务 T 和目标硬件 H，其静态数据布局问题定义如下。

定义 1　在程序空间$\mathbb{X}_T$中搜索满足优化目标 G 的张量程序 X^*，使得在 H 上解决任务 T 所需的运行时间最短。

根据张量程序自动生成框架所采取的搜索策略（静态数据布局策略）的不同，定义 1 中的程序空间$\mathbb{X}_T$（优化空间）和输出程序 X^* 会有所不同。

对于本文关注的 Ansor 而言，其共有 3 种搜索策略：(1) 不进行布局（no layout，NL）策略；(2) 运行时布局（layout during running，LR）策略；和 (3) 编译时布局（layout during compiling，LC）策略。其中第 3 种策略只能用于静态数据，而前 2 种对于动态数据和静态数据均适用。在 Ansor 中，不同策略使用不同的优化空间$\mathbb{X}_T$（$\mathbb{A}_T$或$\mathbb{S}_T$或$\mathbb{P}_T$），但却使用同一个搜索空间$\mathbb{A}_T$，因为定义于输入计算图上的通用派生规则无法直接生成位于$\mathbb{S}_T$和$\mathbb{P}_T$中的程序。因此，Ansor 只能从$\mathbb{A}_T$中（准确地来说应该是$\mathbb{A}_T - \mathbb{S}_T$）采样程序进行搜索，然后经数据布局变换映射到相应优化空间（$\mathbb{A}_T$或$\mathbb{S}_T$或$\mathbb{P}_T$）上进行测量。这一点体现在优化目标 G 上。

对于策略 NL，定义 1 中的$\mathbb{X}_T$是任务 T 的程序空间$\mathbb{A}_T$，输出程序 X^* 是空间$\mathbb{A}_T$中的程序 A^*。优化目标 G 如式 (1) 所示，其中，$timeof(A,H)$ 表示程序 A 在硬件 H 上的执行时间。

$$\langle \min timeof(A,\ H) \mid A \in \mathbb{A}_T \rangle \tag{1}$$

对于策略 LR，定义 1 中的$\mathbb{X}_T$是任务 T 的布局分离程序空间$\mathbb{S}_T$，输出程序 X^* 是空间$\mathbb{S}_T$中的程序 $(L+P)^*$。优化目标 G 如式 (2) 所示，其中，$LT0(A)$ 表示对程序 A 进行数据布局优化得到的布局变换程序，$LT1(A)$ 表示对程序 A 进行数据布局优化得到的布局后程序。

$$\left(\min(timeof(L,H) + timeof(P,H)) \;\middle|\; \begin{array}{c} L = LT0(A) \\ P = LT1(A) \\ A \in \mathbb{A}_T \\ (L+P) \in \mathbb{S}_T \\ P \in \mathbb{P}_T \end{array} \right) \tag{2}$$

对于策略 LC，定义 1 中的$\mathbb{X}_T$是任务 T 的布局后程序空间$\mathbb{P}_T$，优化目标 G 如式 (3) 所示，搜索的输出程序是空间$\mathbb{P}_T$中的程序 P^*。

$$\left(\min timeof(P,H) \;\middle|\; \begin{array}{c} P = LT1(A) \\ A \in \mathbb{A}_T \\ P \in \mathbb{P}_T \end{array} \right) \tag{3}$$

对于策略 LR 和策略 LC，由于优化空间$\mathbb{X}_T$和搜索空间$\mathbb{A}_T$并非同一空间，因此其优化做法——依据在优化空间$\mathbb{X}_T$上定义的优化目标 G 去采样位于搜索空间$\mathbb{A}_T$中的程序 A，以求 A 经过变换后得到的程序 X 能有最小的运行时间 $timeof(X,\ H)$——是无法达到目标的。尤其是对策略 LC，因为其优化空间$\mathbb{P}_T$与搜索空间$\mathbb{A}_T$中程序的访存特征差异过大。

3　采用自适应静态数据布局策略的 AL-Ansor

3.1　Ansor 的工作原理及静态数据布局策略

Ansor 是一种深度学习张量程序自动生成框架，其工作原理如图 4 所示。

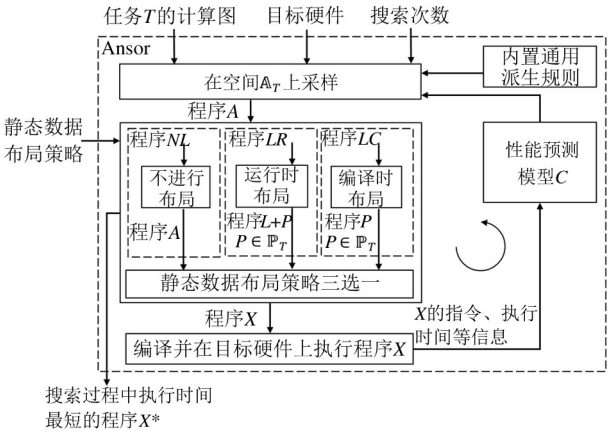


图 4　Ansor 工作原理示意图

Ansor 以待优化算子（记为任务 T）的计算图、目标硬件信息、搜索次数以及指定的某一静态数据布局策略（NL、LR 或 LC）为输入，经过指定次数的搜索后，输出在目标硬件上执行时间最短的张量程序 X^*。在每一轮搜索过程中，Ansor 主要分为 4 步进行：(1) 程序采样。依据性能预测模型 C，从由输入计算图和图上的通用派生规则共同定义的张量程序

空间$\mathbb{A}_T$中采样若干程序。(2)静态数据布局。依据输入的静态数据布局策略,对上一步采样出的每一个程序 A 进行相应的变换,得到程序 X。(3)编译并执行。将程序 X 编译至目标硬件平台,并将相应二进制指令部署至目标硬件平台上运行,这个过程中会得到程序 X 的指令、运行时间等信息。(4)训练性能预测模型 C。从上一步的编译及运行信息中提取特征训练性能预测模型 C,用来指导下一轮搜索过程中第一步的程序采样。

在 Ansor 的一次完整搜索流程中,需要指定固定的一种静态数据布局策略。不同的静态数据布局策略会影响性能预测模型 C 的训练。

对于策略 NL,程序 X 就是采样程序 A。该策略下,性能预测模型 C 以程序 A 的信息来训练。此时 Ansor 的搜索空间和优化空间均为$\mathbb{A}_T$,优化目标为 2.2 节中的式(1)。

对于策略 LR,程序 X 是采样程序 A 经过静态数据布局变换而产生的布局变换程序 L 和布局后程序 P 的组合。该策略下,性能预测模型 C 以$(L+P)\in\mathbb{S}_T$的特征信息训练,来进行程序$A\in\mathbb{A}_T$的选择。此时 Ansor 的搜索空间为$\mathbb{A}_T$,而优化空间为$\mathbb{A}_T$的子空间$\mathbb{S}_T$,优化目标为 2.2 节中的式(2)。

对于策略 LC,程序 X 是采样程序 A 经过静态数据布局变换产生的布局后程序 P,因为布局变换程序 L 被指定在编译时执行。该策略下,性能预测模型 C 以$P\in\mathbb{P}_T$的特征信息训练,来进行程序$A\in\mathbb{A}_T$的选择。此时 Ansor 的搜索空间为$\mathbb{A}_T$,而优化空间为访存局部性更好的布局后程序空间$\mathbb{P}_T$,优化目标为 2.2 节中的式(3)。优化空间和搜索空间不在同一个空间,且其中的程序访存特征具有较大的差异。

表 1 列出了以 32 核 Intel Xeon CPU 为目标硬件,以 ResNet-50 的 4 个卷积块(包括输入补 pad、2D 卷积、加偏置以及做 ReLU 激活)为搜索任务, Ansor 分别在 3 种静态布局策略下搜索 1024 次得到的最优程序 X^* 的运行时间。可以看出,单一的策略 NL、LR 和 LC 并不总在所有搜索任务上表现得最好。

表 1　Ansor 的 3 种静态数据布局策略搜索结果

任务(来自 ResNet-50)	NL/ms	LR/ms	LC/ms
conv1	**3.294**	4.097	4.338
res2a-branch1	2.013	**1.674**	2.146
res2a-branch2a	0.424	0.371	**0.367**
res2a-branch2b	3.748	3.200	**3.039**

3.2　自适应静态数据布局策略

从 Ansor 的工作原理可知,其存在以下 2 个问题:

(1)静态数据布局策略需要预先指定,而针对不同的任务,最优的策略不同,因此预先指定的静态数据布局策略往往不是最优的(单一策略非最优)。

(2)对于编译时布局策略 LC,搜索空间$\mathbb{A}_T$与优化空间$\mathbb{P}_T$存在较大差异,主要体现在程序对静态数据的访存特性上,$\mathbb{P}_T$上的程序访存特性更好。由此训练得到的性能预测模型难以找到性能最优的程序(性能预测模型不准确)。

针对这些问题,本文提出自适应静态数据布局策略(AL)。其主要特点是,在搜索过程中不是只使用一种固定的静态数据布局策略,而是根据一个自适应策略选择函数 Select _ Strategy 选择并应用多种静态数据布局策略(NL/LR/LC),来共同训练性能预测模型 C。一种可能的自适应策略是:在进行策略 LC 前,先使用策略 NL 和策略 LR 来训练模型 C。这样不仅有机会探索到不同策略下的最优程序,还可以缩小待搜索空间。之后再采取策略 LC 微调模型 C,此时优化空间和搜索空间的差异不那么显著,使得模型 C 更容易在搜索空间$\mathbb{A}_T$中找到程序 A^*,其对应的布局后程序 P^* 在目标硬件上执行时间最短。

自适应静态数据布局策略 AL 的伪代码描述如算法 1 所示。

算法 1　AL

输入:搜索空间$\mathbb{A}_T$,目标硬件 H,搜索轮数 L,性能预测模型 C,自适应策略选择函数 *Select _ Strategy*

输出:搜索过程中发现的最优布局后程序$P^*\in\mathbb{P}_T$

1. *Min _ T* = Inf, P^* = Nul //初始化 P^* 及对应的时间 *Min _ T*
2. FOR Search _ Idx IN[0, L):
3. 　*A _ Cand* = *Program _ Sample*($\mathbb{A}_T$, H, C)
4. 　*L _ Cand*, *P _ Cand* = *Layout _ Separate*(*A _ Cand*)
5. 　*Strategy* = *Select _ Strategy*(*Search _ Idx*, L, C)
6. 　*A _ Infos*, *L _ Infos*, *P _ Infos* =

```
    Build _ Run(A _ Cand,L _ Cand,P _ Cand, Strategy,H)
7.  Train _ Model(C, A _ Infos, L _ Infos, P _ Infos, Strategy)
8.  Index = Argbest(P _ Infos)
9.  IF P _ Infos[Index]. runtime < Min _ T:
10.     Min _ T = P _ Infos[Index]. runtime
11.     P* = P _ Cand[Index]
12. RETURN P*
```

算法 1 的输入是任务 T 的张量程序空间$\mathbb{A}_T$(也即该算法的搜索空间),目标硬件 H,搜索轮数 L,以及初始性能预测模型 C。输出是在搜索过程遇到的在目标硬件 H 上执行时间最短的布局后程序 P^*。

在每一轮搜索过程中,主要经过了以下过程。

(1)先根据目标硬件 H 和性能预测模型 C 在搜索空间$\mathbb{A}_T$中采样出候选的程序集合 A _ Cand(第 3 行)。

(2)对集合中每个程序进行数据布局变换,得到布局变换程序集合 L _ Cand 和布局后程序集合 P _ Cand(第 4 行)。

(3)根据自适应策略选择函数 Select _ Strategy 确定本轮搜索应使用的静态数据布局策略(第 5 行)。该函数的输入可以有当前的搜索轮数 Search _ Idx、搜索总轮数 L 以及性能预测模型 C 等。该函数的实现可以是一种启发式算法,一种预定义规则,或者是一个预训练的决策树或神经网络模型。

(4)根据本轮搜索的静态数据布局策略 Strategy,对相应的程序进行编译,并部署到目标硬件 H 上执行(第 6 行)。当 Strategy 为 NL 时,Build _ Run 函数只会对 A _ Cand 和 P _ Cand 中的程序进行编译并运行得到相应的编译及运行信息 A _ Infos 和 P _ Infos。对 A _ Cand 编译并运行是为了训练性能预测模型 C,使其能够锁定一个更小的搜索空间。对 P _ Cand 编译并运行是为了发现具有最短执行时间的布局后程序。当 Strategy 为 LR 时,Build _ Run 函数只会对 L _ Cand 和 P _ Cand 中的程序进行编译并运行得到相应的编译及运行信息 L _ Infos 和 P _ Infos。当 Strategy 为 LC 时,Build _ Run 函数只会对 P _ Cand 中的程序进行编译并运行得到相应的编译运行信息 P _ Infos。

(5)根据策略 Strategy 选择有效的编译及运行信息去训练/微调性能预测模型 C(第 7 行)。

(6)根据布局后程序集合 P _ Cand 的编译运行信息 P _ Infos 中的运行时间信息更新搜索过程中发现的最优布局后程序 P^*(第 8 ~ 11 行)。

与 Ansor 所采用的单一数据布局策略相比,AL-Ansor 有 2 个特点:(1)一个任务 T 的不同搜索轮次之间可以使用不同的静态数据布局策略(由 Select _ Strategy 决定);(2)无论何种静态数据布局策略,每一轮都会编译并运行布局后程序集合 P _ Cand。

3.3 基于 AL 策略的 AL-Ansor

将自适应静态数据布局策略 AL 应用于 Ansor 的整个搜索过程中,便得到了改进的 Ansor——AL-Ansor,如图 5 所示。和图 4 的主要差别在于程序采样和编译执行之间的数据布局变换过程。在 AL-Ansor 框架中,静态数据布局策略的确定不再依靠一次性的输入,而由内部的自适应策略选择函数决定。输出总是执行时间最短的布局后的程序 $P^* \in \mathbb{P}_T$,而不是依赖于输入的静态数据布局策略的程序 X^*。

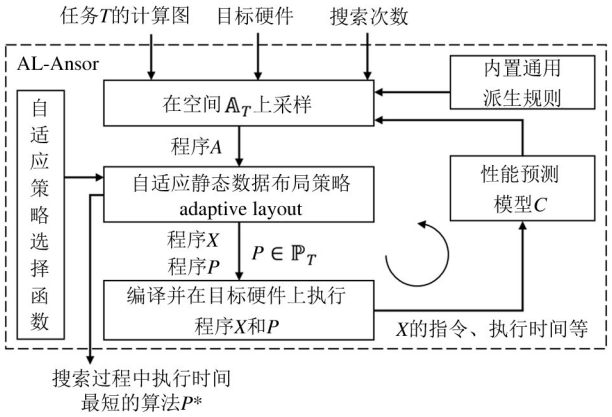


图 5 AL-Ansor 工作原理示意图

4 实 验

4.1 实验设置及任务负载

本文采用的基准 Ansor 框架即 TVM 库(版本为 e718f5a8)的 auto _ scheduler 模块。根据 Ansor 的输入静态数据布局策略的不同,分别将输入 NL、LR 和 LC 策略的 Ansor 记为 NL-Ansor、LR-Ansor 和 LC-Ansor 以方便后续实验展示。Ansor 和 AL-Ansor 搜索框架的目标硬件为 32 核 Intel Xeon Gold 6226R

CPU(@2.90 GHz),搜索次数均设置为 1024 次,因为这一次数已足够使性能预测模型收敛[14]。实验中 AL-Ansor 所采用的自适应策略选择函数 Select _ Strategy 为一种简单的预定义规则:依次进行 NL、LR 和 LC 策略,且 3 种策略搜索次数的占比为 5∶5∶6。

本文采用的任务负载是一些常见于深度卷积神经网络中的卷积块(包含输入补 pad、进行 2D 卷积、加偏置以及做 ReLU 激活),因为卷积是其主要计算负载。这些卷积块用 TVM 的 topi 接口进行描述,然后分别调用 NL-Ansor、LR-Ansor、LC-Ansor 和 AL-Ansor 进行搜索优化。所选卷积任务的输入、输出数据如表 2 所示。

表 2　实验负载

任务	输入(n, hi, wi, ci)	权值(hk, wk, ci, co)	步长	Pad	输出(n, ho, wo, co)
T1	32, 224, 224, 3	7, 7, 3, 64	2	3	32, 112, 112, 64
T2	32, 56, 56, 64	1, 1, 64, 256	1	0	32, 56, 56, 256
T3	32, 56, 56, 64	1, 1, 64, 64	1	0	32, 56, 56, 64
T4	32, 56, 56, 64	3, 3, 64, 64	1	1	32, 56, 56, 64
T5	32, 56, 56, 256	1, 1, 256, 64	1	0	32, 56, 56, 64
T6	32, 28, 28, 128	3, 3, 128, 128	1	1	32, 28, 28, 128
T7	32, 14, 14, 256	1, 1, 256, 1024	1	0	32, 14, 14, 1024
T8	32, 14, 14, 1024	1, 1, 1024, 256	1	0	32, 14, 14, 256
T9	32, 14, 14, 1024	1, 1, 1024, 512	2	0	32, 14, 14, 512
T10	32, 7, 7, 512	3, 3, 512, 512	1	1	32, 7, 7, 512

从 2 个方面对 4 种框架进行比较。一个是不同框架下的搜索时间(或者编译时间,因从计算图到生成张量程序的搜索过程可以视为编译);另一个是它们搜索出的最优程序的运行时间。在进行实验时,独占目标服务器测量时间。对于最优程序运行时间的测量,采取以下方法:对搜索得到的最优程序进行 3 组测量取平均值,每组测量 100 轮取中位数,每轮运行至少 500 ms(根据程序执行时间,每轮的执行次数在几百至几千的范围),以最小化可能存在的测量误差。

4.2　搜索时间对比

图 6 展示了 NL-Ansor、LR-Ansor、LC-Ansor 和 AL-Ansor 的搜索时间对比。其中 AL-Ansor 的搜索时间被拆分为 3 部分:AL-Ansor-base、AL-Ansor-exb 和 AL-Ansor-exr。AL-Ansor-exb 和 AL-Ansor-exr 分别表示 AL-Ansor 的自适应静态数据布局策略 AL 所引入的程序 P 的编译开销和运行开销,因为在搜索过程中无论选取哪一种布局策略,都要生成 P 并对其进行编译和运行,这一点通过对比图 4 和图 5 可以看出。但这一开销只会发生在自适应策略选择函数返回策略 NL 的搜索轮次;对于返回策略 LR 和策略 LC 的搜索轮次,程序 P 本就包含在程序 X 之中,因此不会产生额外的编译开销和运行开销。AL-Ansor-base 所示的时间则为把这部分开销去掉的时间。

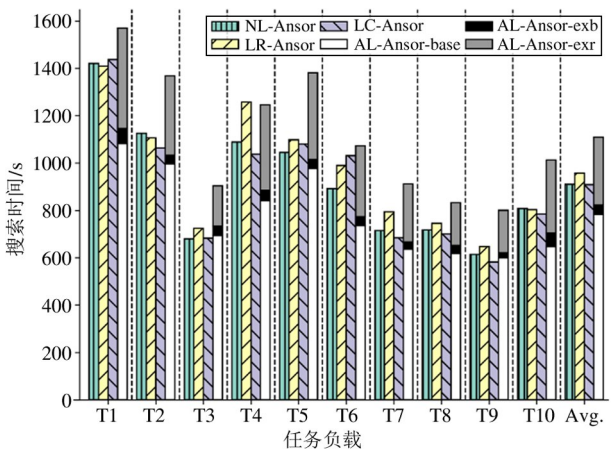


图 6　NL-Ansor、LR-Ansor、LC-Ansor 和 AL-Ansor 的搜索时间对比

从图 6 可以观察到以下现象:

(1) AL-Ansor-base 在 10 个任务上的搜索时间相较于 NL-Ansor、LR-Ansor 和 LC-Ansor 平均提升了 13.86% ~18.24%。这些主要来自于布局变换后的

程序 X 的编译和运行时间提升，而程序 X 是采样程序 A 经过静态数据布局变换得到的，这表明策略 AL 下决定采样质量的性能预测模型 C 要优于其他单一策略。

(2)相较于 AL-Ansor-base，在每个搜索轮次引入对程序 P 的评估(AL-Ansor-exb 和 AL-Ansor-exr)会带来平均 41.82% 的时间开销。如前所述，该部分开销主要是由策略 NL 所在搜索轮次引入的。这部分开销中 87.67% 来自于程序 P 在目标硬件上的运行过程(AL-Ansor-exr)，这是为了估计 P 部署到目标硬件后的真实运行时间，以在搜索过程中选出最优的布局后程序 P^*。AL-Ansor-exr 的开销可以根据需要去调整：如果希望对程序 P 的评估尽可能准确，则可以将这部分时间延长；如果希望整个搜索过程快一些，则可以将部分时间适当缩短。

(3)在 10 个任务上，相较于 NL-Ansor、LR-Ansor 和 LC-Ansor，AL-Ansor 平均上有 15.95% ~22.17% 的搜索时间开销。但是，由于 AL-Ansor 内的静态数据布局策略是自适应选择的，而 Ansor 中无法预先确定哪一种策略下能找到当前任务的最优张量程序。所以，为了能找到最优的张量程序，Ansor 需要尝试每一种策略。从这个角度看，AL-Ansor 的搜索时间相较于 Ansor 平均会有 2.5 倍的搜索时间提升。

总体来看，AL-Ansor 在搜索时间上仍然是优于最优静态数据布局策略未知的 Ansor 的。虽然相较于确定采用某一种静态数据布局策略的 Ansor(NL-Ansor、LR-Ansor 或 LC-Ansor)，AL-Ansor 会有一定的搜索时间开销。但是，对于一个确定的任务而言，搜索过程往往是一次性的，而搜索得到的最优张量程序在部署到目标硬件后则会反复运行使用。因此部署后运行时间的提升更为关键，部署前搜索时间的少量开销是可以接受的。下面将分析 4 种自动生成框架得到的最优张量程序运行时间。

4.3　最优程序运行时间对比

图 7 中依次展示了由 NL-Ansor、LR-Ansor 和 AL-Ansor 搜索出的最优张量程序 X，相对于 LC-Ansor 搜索出的最优张量程序 Y 的性能提升百分比 y，即 y 由式(4)得到：

$$y = \frac{timeof(Y,H) - timeof(X,H)}{timeof(Y,H)} \times 100 \quad (4)$$

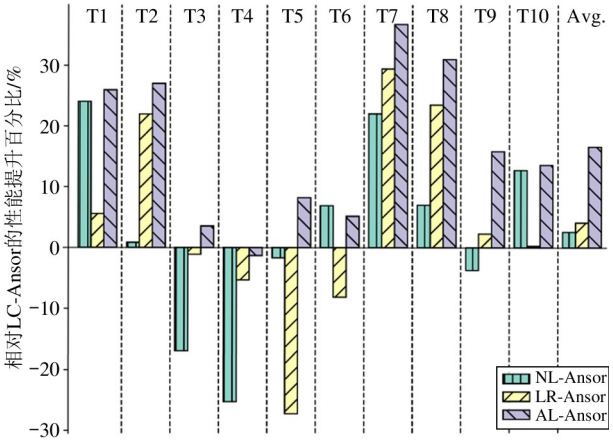


图 7　NL-Ansor、LR-Ansor 和 AL-Ansor 相对于 LC-Ansor 的性能提升

其中 $timeof(Y, H)$ 为程序 Y 在硬件 H 上的执行时间。

在图 7 中，条形图所处方位展示了对应框架采取策略与 LC 的优劣：位于 x 轴上方表示对应策略优于策略 LC，位于 x 轴下方则表示劣于策略 LC。条形图的高度则展示了对应策略与 LC 的差距：高度越大，表示对应策略所得最优程序与策略 LC 所得最优程序之间的运行时间绝对值之差越大。从图 7中可以得出如下结论。

(1)Ansor 原本支持的 3 种独立策略 NL、LR 和 LC 并没有绝对的优劣之分。这再一次表明了由用户预先指定单一的静态数据布局策略是不恰当的。表 3 展示了 3 种策略的不同优劣排序在任务负载中均曾出现。

表 3　3 种策略优劣排序在 T1 ~ T10 上的发生情况

策略优劣排序	所在任务
NL > LR > LC	T1, T10
NL > LC > LR	T6
LR > NL > LC	T2, T7, T8
LR > LC > NL	T9
LC > LR > NL	T3, T4
LC > NL > LR	T5

注：S1 > S2 > S3 表示 S1 优于 S2，同时 S2 优于 S3

(2)策略 NL、策略 LR 和策略 AL 相较于 LC 分别平均提升了 2.59%、4.13% 和 16.59% 的性能。

策略 NL 和策略 LR 相较于策略 LC 其性能平均上有提升,这一点虽然反直觉,但也进一步印证了在策略 LC 下,搜索空间$\mathbb{A}_T$和优化空间$\mathbb{P}_T$之间的差异对于张量程序搜索框架的不利影响。

(3)策略 AL 在绝大多数任务上优于其他策略。只在任务 T4 和 T6 上存在 2 处例外,在 T4 上,策略 AL 相对于 LC 损失了 1.88% 的性能;而在 T6 上,AL 相对于 NL 损失了 1.31% 的性能。这些微小的损失可能来自于搜索过程中为了跳出局部最小点的探索机制,而此过程是随机不可控的。但值得注意的是,由此为策略 AL 带来的影响是很小的。除去这 2 个异常点,策略 AL 相对于策略 NL 提升了 0.98% ~ 26.40%(T10 上最小,T2 上最大,见图 8),相对于策略 LR 提升了 3.79% ~27.91%(T4 上最小,T5 上最大,见图 8),相对于策略 LC 提升了 3.58% ~36.80%(T3 上最小,T7 上最大)。

图 8 单独展示了 AL-Ansor 相对于 NL-Ansor 和 LR-Ansor 的性能提升百分比。在所选的 10 个任务负载上,策略 AL 相对于 NL 平均提升了 13.81%,相对于 LR 平均提升了 12.41%。

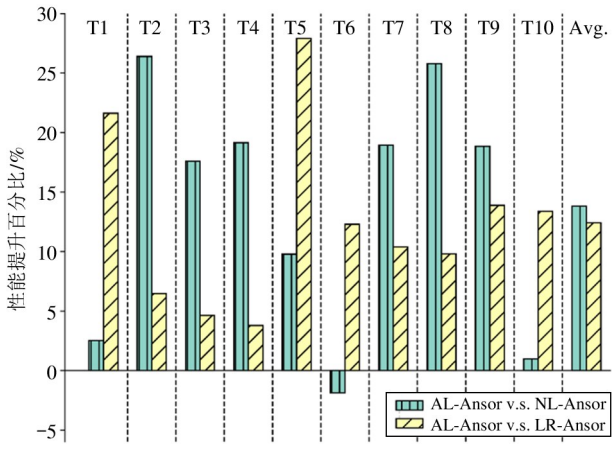


图 8　AL-Ansor 相对于 NL-Ansor 和 LR-Ansor 的性能提升

综合来看,相较于使用单一静态数据布局策略的 Ansor,采用自适应静态数据布局策略的 AL-Ansor 框架更加合理和高效。

5　结　论

本文研究了静态数据布局在深度学习张量程序自动生成框架中的应用。发现应用最广泛、前景最佳的 Ansor 框架存在 2 个问题:(1)只能指定单一的静态数据布局策略(NL 或 LR 或 LC)进行张量程序的搜索,而单一布局策略具有非最优问题;(2)对于专用于静态数据布局的策略 LC,其搜索空间和优化空间因其中程序的访存特征差别而存在较大差异,导致性能预测模型不准确的问题。这些问题都会导致搜索得到的张量程序仍有较大优化空间。针对这些问题,本文提出了自适应的静态数据布局策略 AL 及基于此策略的深度学习张量程序自动生成框架 AL-Ansor。实验表明,在张量程序搜索过程中,自适应地使用多种不同的静态数据布局策略,不仅能够缓解如何选择合适布局策略的难题,也能够减小搜索空间和优化空间的差异,因此搜索生成的张量程序具有更好的性能。这为深度学习的高效部署与应用带来了广泛的好处。

虽然本文讨论的主要对象是静态数据,目标硬件平台选取的是 CPU,但对于动态数据或者其他目标硬件平台(如 GPU、TPU 等),本文所揭示的原理也是适用的。另外,本文所提出的自适应策略选择函数是决定该方法好坏的一个因素,如何选择它仍是有待研究的课题。

参考文献

[1] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[C]// Advances in Neural Information Processing Systems. Lake Tahoe: NIPS, 2012:1097-1105.

[2] 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.

[3] SABOUR S, FROSST N, HINTON G E. Dynamic routing between capsules[C]// Advances in Neural Information Processing Systems. Long Beach: NIPS, 2017:3856-3866.

[4] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is all you need[C]// Advances in Neural Information Processing Systems. Long Beach: NIPS, 2017:5998-6008.

[5] CHEN T, DU Z, SUN N, et al. Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM SIGARCH Computer Architecture

News, 2014,42(1):269-284.

[6] CHEN Y, CHEN T, XU Z, et al. DianNao family: energy-efficient hardware accelerators for machine learning [J]. Communications of the ACM, 2016,59(11):105-112.

[7] JOUPPI N P, YOUNG C, PATIL N, et al. In-datacenter performance analysis of a tensor processing unit[C]// Proceedings of the 44th Annual International Symposium on Computer Architecture. Toronto: IEEE, 2017:1-12.

[8] HAN S, KANG J, MAO H, et al. ESE: efficient speech recognition engine with sparse LSTM on FPGA[C]// Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Monterey: Association for Computing Machinery, 2017:75-84.

[9] PARK G, IM D, HAN D, et al. A 1.15 TOPS/W energy-efficient capsule network accelerator for real-time 3D point cloud segmentation in mobile environment [J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2020,67(9):1594-1598.

[10] 许浩博, 王颖, 王郁杰, 等. 面向深度可分离卷积的硬件高效加速器设计[J]. 高技术通讯, 2021,31(8):791-799.

[11] CHEN J, RAN X. Deep learning with edge computing: a review[J]. Proceedings of the IEEE, 2019,107(8):1655-1674.

[12] VÉSTIAS M P, DUARTE R P, DE SOUSA J T, et al. Moving deep learning to the edge[J]. Algorithms, 2020,13(5):125.

[13] LI E, ZENG L, ZHOU Z, et al. Edge AI: on-demand accelerating deep neural network inference via edge computing[J]. IEEE Transactions on Wireless Communications, 2019,19(1):447-457.

[14] ZHENG L, JIA C, SUN M, et al. Ansor: generating high-performance tensor programs for deep learning[C]// Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. Online: USENIX, 2020:863-879.

[15] CHEN T, MOREAU T, JIANG Z, et al. TVM: An automated end-to-end optimizing compiler for deep learning [C]// Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad: USENIX, 2018:578-594.

[16] QIU K, CHEN W, XU Y, et al. A peripheral circuit reuse structure integrated with a retimed data flow for low power RRAM crossbar-based CNN[C]// Design, Automationand Test in Europe Conference and Exhibition. Dresden: IEEE, 2018:1057-1062.

[17] LEE J J, LI P. Reconfigurable dataflow optimization for spatiotemporal spiking neural computation on systolic array accelerators[C]// IEEE 38th International Conference on Computer Design. Hartford: IEEE, 2020:57-64.

[18] VANHOUCKE V, SENIOR A, MAO M Z. Improving the speed of neural networks on CPUs[C]// Deep Learning and Unsupervised Feature Learning Workshop of Advances in Neural Information Processing Systems. Granada: NIPS, 2011:1-8.

[19] YU C, ROY P, BAI Y, et al. Lwptool: a lightweight profiler to guide data layout optimization [J]. IEEE Transactions on Parallel and Distributed Systems, 2018,29(11):2489-2502.

[20] DONG X, LIU L, ZHAO P, et al. Acorns: a framework for accelerating deep neural networks with input sparsity [C]// Proceedings of the 28th International Conference on Parallel Architectures and Compilation Techniques. Seattle, 2019:178-191.

[21] CHEN T, ZHENG L, YAN E, et al. Learning to optimize tensor programs[C]// Advances in Neural Information Processing Systems. Montréal: NIPS, 2018:3389-3400.

[22] ZHENG S, LIANG Y, WANG S, et al. Flextensor: an automatic schedule exploration and optimization framework for tensor computation on heterogeneous system[C] // Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ASPLOS, 2020:859-873.

[23] JIA Y, SHELHAMER E, DONAHUE J, et al. Caffe: convolutional architecture for fast feature embedding[C] // Proceedings of the 22nd ACM International Conference on Multimedia. New York: Association for Computing Machinery, 2014:675-678.

[24] GEORGANAS E, AVANCHA S, BANERJEE K, et al. Anatomy of high-performance deep learning convolutions on simd architectures[C]// International Conference for High Performance Computing, Networking, Storage and Analysis. Dallas: IEEE, 2018:830-841.

[25] DU W, WU L, CHEN X, et al. ZhuQue: a neural network programming model based on labeled data layout

[C]// International Symposium on Advanced Parallel Processing Technologies. Tianjin: APPT, 2019:27-39.

[26] 吴林阳,杜伟健,陈小兵,等. 一种运算和数据协同优化的深度学习编译框架[J]. 高技术通讯,2020,30(2):120-125.

[27] LIU Y, WANG Y, YU R, et al. Optimizing CNN model inference on CPUs[C]//2019 USENIX Annual Technical Conference. Renton: USENIX, 2019:1025-1040.

A deep learning tensor program automatic generation framework based on adaptive layout of static data

FAN Zhe[*,**], NAN Ziyuan[*,**], HAO Yifan[*], DU Zidong[*], CHEN Yunji[*,**]

([*]State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

How to determine the layout of static/const data is a big challenge faced by tensor program automatic generation frameworks. Ansor, the most broadly-used and promising framework among them, solves this issue by training a performance cost model according to a layout strategy specified in advance, then searching the tensor program with the optimal performance based on the cost model. However, there are two problems: a single strategy cannot be suitable for all tasks, and the performance cost model is not accurate. In order to solve these problems, AL-Ansor, a tensor program automatic generation framework based on the adaptive layout (AL) strategy of static data, is proposed. It adaptively chooses multiple layout strategies during the search process, and trains the performance cost model according to them. In this way, AL-Ansor can find a tensor program with higher performance. Taking convolutional layers as workloads, this work evaluates Ansor and AL-Ansor in a target server with a 32-core Intel Xeon CPU. The experimental results show that AL-Ansor improves the execution performance by 13.81%, 12.41%, and 16.59%, respectively, on average, compared against Ansor with three specified layout strategies.

Key words: deep learning, tensor program automatic generation framework, layout of static/const data, adaptive strategy, performance cost model