计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(2):220−226 [doi: 10.15888/j.cnki.csa.008320]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

异构并行平台的 Caffe 推理速度提升方法①

王子曦, 邵培南, 邓　畅

(中国电子科技集团公司第三十二研究所, 上海 201808)
通信作者: 王子曦, E-mail: 10152130146@stu.ecnu.edu.cn

摘　要: 随着计算机硬件性能的提高, 目前在个人终端上也开始出现使用预训练机器学习模型进行推理的运用. Caffe 是一款流行的深度学习框架, 擅长图像分类等任务, 但是在默认状态下只能单核运行, 无法充分发挥异构并行计算设备的计算能力. 深度学习对于计算性能的要求较高, 如果能并行化以充分使用所有计算设备, 就能提升计算速度和使用体验. 由于 CPU 和 GPU 的计算性能之比在不同模型下存在差异, 因此不能简单将任务均分到多个计算设备. 而任务拆分过多或者需要等待多设备完成任务后同步的调度算法会引入更多开销. 因此, 还需要设计合适的调度算法减少设备空闲时间, 才能获得更好的性能. 已有一些提高 Caffe 并行表现的方法, 但是对于具体平台有限制且使用难度较高, 无法简单充分利用异构并行计算设备的计算能力. 本文将 Caffe 接口扩展, 使得自定义程序可以调用异构并行平台的多核或多计算设备使用 Caffe 进行深度学习推理. 接着将目前已有的多种调度算法运用到此类任务上并考察了运行效果. 为了减少已有调度算法的同步开销, 本文提出了先进先出调度和快速分块调度两种新的算法. 测试表明, 使用快速分块调度算法结合异构并行计算设备, Caffe 的推理速度相比只使用单个 CPU 核心或者单个 GPU 都大幅提升. 而且, 相比已有调度算法中表现最好的 HAT 算法, 本文提出的快速分块调度算法在 MNIST 和 Cifar-10 两个数据集上分别减少了 7.4% 和 21.0% 的计算性能浪费.

关键词: 调度算法; Caffe 推理加速; 快速分块调度算法; 异构并行平台调度; 深度学习性能优化

引用格式: 王子曦,邵培南,邓畅.异构并行平台的 Caffe 推理速度提升方法.计算机系统应用,2022,31(2):220–226. http://www.c-s-a.org.cn/1003-3254/8320.html

Caffe Inference Acceleration Method on Heterogeneous Parallel Platform

WANG Zi-Xi, SHAO Pei-Nan, DENG Chang

(The 32nd Research Institute of China Electronics Technology Group Corporation, Shanghai 201808, China)

Abstract: With the development of computer performance, pre-trained machine learning models are used for inference on personal devices. Caffe is a popular deep learning framework featuring image classification. However, it can only infer using one CPU core or one GPU if without customization, which limits the computing power of heterogeneous parallel computation devices. Deep learning is a demanding task for a computation device. For a better user experience and faster inference, it is important to fully use all computing cores of the device via parallelization. Considering the CPU-to-GPU performance ratio may vary on different deep learning models, tasks should not just be equally assigned to all computing cores. It should be noted that more overhead will be introduced if the tasks are divided into too many portions or synchronized scheduling algorithms are used. Thus, a well-designed scheduling algorithm able to reduce idle time is crucial for better performance. Some approaches have been developed to improve Caffe performance on heterogeneous parallel computation devices, whereas there are some limits on the platform hardware and usage. As a result, it is difficult to fully utilize the performance of these devices. This study reports the work on the improvement of Caffe interface and

① 收稿时间: 2021-04-14; 修改时间: 2021-05-11, 2021-05-27; 采用时间: 2021-06-02; csa 在线出版时间: 2022-01-17

the proposed new algorithms. Caffe interface is extended to enable customized programs to use multiple computing cores or devices of a heterogeneous parallel platform for deep learning inference with Caffe. Some existing scheduling algorithms are ported and tested. To avoid synchronization overhead, two novel asynchronous scheduling algorithms, async-FIFO and fast-split, are proposed. All scheduling algorithms are tested and results show that the Caffe inference performance of heterogeneous parallel computation devices adopting fast-split is significantly faster than that in the case where only one computing core is adopted. Fast-split on average reduces performance waste by 7.4% and 21.0% on MNIST and Cifar-10 datasets, respectively, compared with the current best heterogeneous parallel scheduling algorithm HAT.

Key words: scheduling algorithm; Caffe inference acceleration; fast-split scheduling algorithm; scheduling on heterogeneous parallel platform; deep learning performance optimization

1 引言

随着计算机硬件性能提升, 多核心 CPU 已经成为主流, 越来越多云服务器开始使用 GPU 加速机器学习等计算任务. 性能较高的个人计算机一般是多核 CPU+单个独立 GPU 的模式. 依靠计算机性能的发展, 擅长图形相关机器学习任务的深度学习发展迅速, 不仅在云服务器上作为 API 提供, 在用户客户端上也已出现使用预训练模型进行数据处理的需求和实现[1], 例如相册根据图片内容进行分类、相机检测拍摄场景类型等[2].

由于深度学习一般运算量大且处理的数据多是图像等复杂数据, 所以有必要让程序充分并行化且尽可能利用硬件加速, 来提升运算速度和用户使用体验. 为了提高硬件利用率, 需要同时使用 CPU 的多个核心和 GPU 执行任务. 由于在实际应用中 GPU 在执行不同类型任务时相对 CPU 的速度并不成固定比例, 如果将任务简单地按照固定比例分配给 CPU 和 GPU, 无法在不同类型的任务上均能充分利用硬件, 所以需要使用恰当的任务调度算法在运行时进行任务分配以均衡硬件的负载.

Caffe[3] 是一款使用广泛的深度学习框架, 主要面向图像分类任务, 主要优点是速度极快. 在目前的主流深度学习框架中, Caffe 的速度、编程工作量、稳定性比较均衡[4], 而且模块化设计便于调用和扩展. 通过部署已经提前训练好的 Caffe 模型, 可以在毫秒级对输入的新数据完成推理. Caffe 的使用流程是先用已标注的数据进行较慢的训练, 然后使用训练结果进行较快的推理. 在实际使用中, 生产环境和普通用户终端一般都是使用预训练模型执行推理任务. 如果已经有训练好的 Caffe 模型, 需要使用 Caffe 的 Python 或者 C++、Matlab 接口[5] 编写程序, 然而在默认情况下只能使用单个 GPU 或者单个 CPU 核心进行推理. 在使用 BLAS 编译 Caffe 后可以提升多 CPU 核心上的并行效果. 不过直接部署 Caffe 无法简单同时使用多核心 CPU 和 GPU 甚至多 CPU、GPU 加速推理速度.

为了提高 Caffe 在异构并行计算设备上的表现, 目前有一些提高 Caffe 并行表现的研究. 例如英特尔的修改版 Caffe 可以在多处理器多核多线程 (NUMA 架构) CPU 计算平台, 尤其是使用特定 Xeon Phi 处理器的平台上大幅提升 Caffe 的性能[6]. 英伟达的 gpu-rest-engine 项目[7] 可以让使用英伟达 GPU 进行运算的 Caffe 提供低延迟的 REST API 图像分类微服务. 还有 AMD 修改的 OpenCL 版 Caffe[8], 经过 AMD 优化并测试, 在 AMD 的 CPU、GPU、APU 等多核异构的计算设备上训练和推理的速度都有提升, 且训练速度的提升倍数更大[9]. 还有结合特定平台上的编译工具进行编译并扩展到分布式服务器来提高 Caffe 运算速度的研究[10]. 这些提高 Caffe 并行表现的方法对于平台有一定限制, 且很多需要自行编译, 使用难度较高, 无法简单充分利用异构并行计算硬件的计算性能.

为了使部署的 Caffe 能充分使用异构并行计算平台上的所有计算设备, 本文做了以下工作.

(1) 为了使部署的 Caffe 调用更灵活, 本文封装了 Caffe 的部署程序接口[11], 并扩展了功能, 使之可以在初始化运行后通过本地环回提供服务, 接收并根据预训练模型执行推理任务.

(2) 分析并总结了现有的几种异构调度算法, 将这些算法应用到在异构并行计算平台上加速 Caffe 执行典型图像分类任务. 通过分析实验数据和观察已有调

度算法的不足之处, 提出了两种新的调度算法, 并进行了相应的实验和数据分析.

2 目前已有调度算法

对于并行化的任务, 任务调度算法决定了负载均衡的效果和各个计算节点的利用率. 由于 GPU 和 CPU 执行不同任务的速度之比并非固定且 GPU 性能随着负载变化性能会有微小变化[12], 还考虑到任务调度、同步带来的开销, 需要合适的任务调度算法才能进一步提高模型的推理速度. 当前使用的几种主流调度算法[13] 如下.

2.1 静态调度

静态调度是一种简单的调度算法, 此算法直接将任务按照固定比例分配到不同的运算核心. 假设总任务数量为 W, 用户指定的 n 个计算设备工作量比例 $r=[r_1,r_2,\cdots,r_n]$, 则第 i 个计算设备在开始时分配任务数量w_i为:

$$w_i = W\cdot\frac{r_i}{\sum_i r_i} \tag{1}$$

由于通常无法知道各个设备在未知任务上的计算能力, 所以只能将任务均匀分配给各个设备, 此时第 i 个计算设备在开始时分配的任务数量w_i为:

$$w_i = \frac{W}{n} \tag{2}$$

由于 CPU 核心和 GPU 之间的运算速度比例在不同运算任务上不同, 所以不可能存在一个固定的比例在所有运算任务上都能自始至终充分利用所有核心.

2.2 快速调度

考虑到静态调度无法根据不同运算核心的性能区别分配任务, 快速调度将任务执行分成以下两步.

第 1 步为小规模测试, 给所有计算核心分配相同的数量较小的任务, 并等待这些核心执行完毕.

第 2 步为正式执行, 根据之前第 1 步各个核心的执行时间, 可以计算出各个核心的执行速度. 根据核心的速度之比将剩下的所有任务按照这个比例分配给相应核心.

假设第 1 步小规模测试得出各个设备的计算性能 $v=[v_1,v_2,\cdots,v_n]$, 此时剩余任务数量为$w_{剩余}$, 则第 2 步正式执行中第 i 个设备分配到的任务数量w_i为:

$$w_i = w_{剩余}\cdot\frac{v_i}{\sum_i v_i} \tag{3}$$

这种调度方法优点是任务没有划分很多, 所以引入的额外开销很小. 缺点是第 1 步分配的任务数量会影响到最终效果. 如果分配过少, 偶然误差更大, 更有可能因为速度估测不准导致第 2 步分配效果不好; 如果分配过多, 虽然估测更加精准, 但是等待第 1 步结束的同步时间更长, 浪费了速度更快的计算设备的计算能力.

2.3 分片调度

分片调度将任务分成等量的小块, 第一次将小块任务等量分配给各个计算单元. 之后每一次等待前一次所有运算完成得出运算速度后按照相应比例分配后续小块.

假设某次同步时得出上一轮各个设备的计算性能 $v=[v_1,v_2,\cdots,v_n]$, 每一个小块包含的任务数量为 w, 则第 i 个设备分配到的任务数量w_i为:

$$w_i = w\cdot\frac{v_i}{\sum_i v_i} \tag{4}$$

此算法优点是对于各个计算设备的计算能力估测更准确. 缺点是如果小块偏小, 等待计算完毕同步的时间会积少成多; 如果小块偏大, 刚开始各个设备之间任务完成的时间差距更大, 导致计算能力的浪费.

2.4 HAT 调度

HAT 调度相比分片调度, 在每一次分片时考虑上一次的任务执行情况, 如果上一次各个计算设备的任务执行时间已经很接近或者剩下任务不够多, 就直接把剩余所有任务按照比例分配, 否则把分片大小扩大一倍按照比例分配.

此方法的优点是减少了同步开销, 还避免了开始时各个计算设备计算能力的差异导致完成时间差距过大.

3 算法设计

目前已有的快速调度、分片调度、HAT 调度算法, 由于存在等待各个计算设备任务完成来统计计算性能这一同步步骤, 会带来开销. 为减少同步开销, 本文提出以下两种无需等待同步的调度算法.

3.1 先进先出调度

此调度算法设计思想来自常见的先进先出算法. 将固定任务数量的小块分配给各个计算设备. 假如用户定义的小块大小是 c, 则第 i 个设备分配到的任务数

量w_i为:

$$w_i = c \tag{5}$$

如果某个设备计算完毕则立即再分配任务小块, 直到所有任务完成. 此方法没有等待各个设备同步的开销, 能充分利用所有设备的计算能力.

先进先出调度中的小块始终大小相等, 对于前期来说分片过多会造成更多开销, 而在最后结尾部分很容易各个设备结束时间差别过大. 因此, 本文进一步提出下面的快速分块调度算法.

3.2 快速分块调度

此算法同时吸收了分片算法估测计算设备性能准确的优点和 HAT 分块数量少的优点. 首先将固定的任务小块w_s分给各个计算设备, 如果某设备完成了小块任务但此时不是所有设备都完成过至少一次小块任务, 就再次给该设备相同任务数量的小块, 直到所有设备都至少完成过一次小块任务. 每次小块任务完成, 都统计出该设备的计算性能v_i. 之后按照设备计算性能给每个空闲设备分配剩余任务的固定比例. 考虑到后期任务会分配过少, 所以剩余任务少于 100 时则分配全部剩余任务. 假设已得出所有设备的计算性能$v = [v_1, v_2, \cdots, v_n]$, 固定比例为 r, 当前剩余工作量为$w_{剩余}$. 则给空闲设备分配的任务数量为:

$$\begin{cases} w_i = w_{剩余} \cdot r \cdot \dfrac{v_i}{\max(v)}, & w_{剩余} \geqslant 100 \\ w_i = w_{剩余}, & w_{剩余} < 100 \end{cases} \tag{6}$$

每个计算设备的程序流程如图 1 所示.

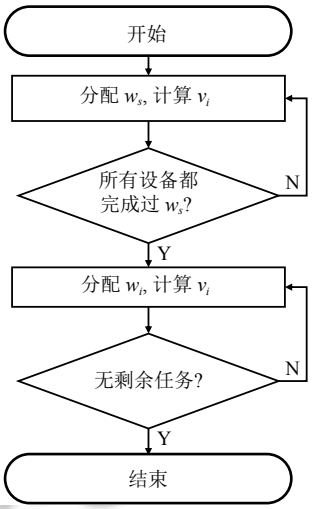

图 1 快速分块调度流程图

4 接口扩展

Caffe 的使用流程是先用已标注的数据进行较慢的训练, 然后使用训练结果进行较快的推理. 在实际使用中, 生产环境和普通用户终端需求的多是推理任务. Caffe 默认只能使用 GPU 或者单个 CPU 进行推理. 为了解决重复初始化 Caffe 带来的开销和提高 Caffe 推理对异构并行硬件的利用率, 本文编写了封装的 Caffe 部署程序 (后续简称服务端), 提供简洁高效的调用接口且易于部署.

为了能够一次初始化 Caffe 并载入预训练模型后多次推理, 且可以通过程序分配任务便于后续使用自定义调度算法, 采用了本地环回网络通信, 服务端通过 UDP 和调用程序通信. 通信时通过 UDP 发送 JSON 数据, 数据可以包括执行的命令和参数等, 便于扩展.

为了使服务端的部署更加便捷, 使用了 Caffe 的 Docker 版. 运行时, 可以通过 Docker 启动参数直接使用某个特定的 CPU 核心[14]. 由于即使使用 GPU 版进行 Caffe 推理也需要 CPU 分配任务处理数据, 会大量占用一个 CPU 核心, 所以服务端需要预留一个 CPU 核心用于工作在 GPU 模式的 Caffe.

5 实验测试和分析

使用本文编写的服务端, 自定义程序可以便捷地调用 Caffe 进行推理. 本节给出有关测试结果, 实验环境配置参数列于表 1.

实验使用 MNIST[15] 和 Cifar-10[16] 这两个经典的图像分类任务数据集. MNIST 数据集为单通道灰度 28×28 图片, Cifar-10 数据集为三通道彩色 32×32 图片. 测试时这两个数据集均预处理为 Caffe 推理支持的 numpy 格式的.npy 文件且载入内存, 避免磁盘读取缓存带来的误差.

表 1 实验环境配置表

环境	参数
CPU	Intel i5 4460 4核心4线程 3.4 GHz
GPU	Nvidia GTX1050 2 GB显存
内存	8 GB×2 DDR3 1 600 MHz
CUDA 驱动	CUDA版本11.2 驱动版本460.39
操作系统	Ubuntu 18.04 64位

首先, 测试了不使用并行化而只使用 CPU 单核或者单个 GPU 在 MNIST 和 Cifar-10 数据集上进行推理的速度, 总任务数量为 10 000, 得到的每秒执行推理任务数列在表 2 中.

表 2 每秒执行的推理任务数

每次提交任务数	MNIST		Cifar-10	
	CPU单核	GPU	CPU单核	GPU
10 000	1 191.9	2 714.4	397.2	2 475.2
1 000	1 187.2	2 711.7	396.9	2 448.4
100	1 181.9	2 568.2	397.0	2 441.1
10	1 140.1	2 441.5	393.1	2 195.5
1	953.8	1 798.6	364.2	1 652.9

结果表明, 每次提交的任务数过少会因为开销导致性能下降. 而且, 对于不同任务, CPU 和 GPU 的性能之比不是恒定的.

根据表 2 数据, 得到 Caffe 推理速度与每次提交任务数的变化关系, 见图 2. 可以看到, 无论是 CPU 和 GPU, 如果每次提交任务数偏少都会因为开销导致性能下降, 且 GPU 下降更明显. 从图 2 中可以看出, 如果任务拆分到 1 000 个左右作为一次提交, 对性能的影响不大, 如果小于 100 个左右, 影响就较大, 出现性能明显下降. 因此, 后面实验测试异构并行计算平台的 Caffe 推理调度算法时, 避免将任务拆分到小于 100 个任务的小块.

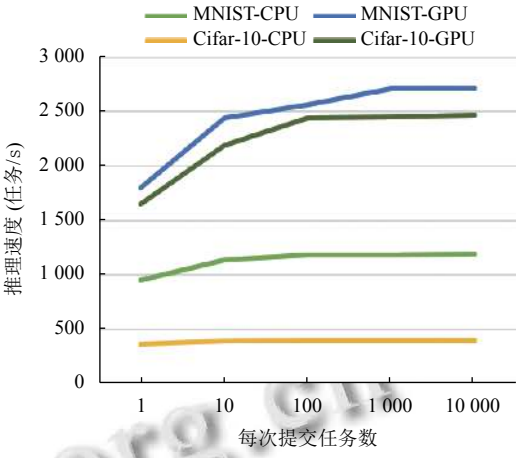

图 2 Caffe 推理速度和每次提交任务数的关系图

接下来在异构并行平台测试了几种调度算法对 MNIST 和 Cifar-10 数据集上的 Caffe 推理任务的加速效果. 考虑到某些算法的参数可变, 实验中针对这些参数的意义, 在其合理范围内取了多个典型值, 以测试不同参数下的运行效果. 不同算法中可变参数的含义和取值列于表 3.

表 3 调度算法的参数和实验取值

算法	参数含义	实验取值
快速调度	第一步各个核心分配任务数	250、500、1 000、2 000
分片调度	每个小块的任务数w	1 000、2 000、5 000
HAT调度	初始时的小块大小s	500、1 000、2 000
先进先出调度	每个小块的任务数	500、1 000、2 000、3 000
快速分块调度	最快核心获得的任务比例r	25%、33.3%、40%、50%

分别使用上文中的各种调度算法及按表 3 的参数取值, 在 MNIST 和 Cifar-10 数据集上进行 Caffe 推理(测试的任务数均为 100 000) 并计时, 根据完成任务所用时间计算出每秒推理数, 即推理速度, 总结在图 3 和图 4 中. 图中同一调度算法从左到右的几个推理速度分别对应表 3 中从左到右的参数取值. 为了便于比较, 同样任务数下只使用 CPU 单核和单个 GPU 的推理速度也画在图中.

从图 3 和图 4 都能看到, 使用能在运行时根据运算设备性能动态分配任务的调度算法, 例如快速调度、分片调度、HAT 调度、先进先出调度、快速分块调度算法, 相比不将推理过程并行化或者简单地静态分配, 都能大幅提升推理速度. 实验中发现, 计算机的各个计算设备占用率基本维持在满载, 且各个设备任务完成的时间差异不大, 说明上述能在运行时动态分配任务的调度算法都成功使用异构并行设备的计算能力, 提高了 Caffe 推理的速度.

图 3 和图 4 也给出了对于同一算法改变可调参数后的推理速度的变化. 可以看出, 对于分片调度、HAT 调度和快速分块调度这 3 种算法, 参数改变产生的差异更小. 分析其算法原理可以知道, 这几种算法通过更多分片能更准确地度量各个计算设备的当前计算能力, 从而更加准确地分配任务. 相比之下, 快速调度由于只通过一次小规模测试来度量各个计算设备的计算能力, 无法适应随着计算设备负载变化而出现的性能变化. 先进先出调度由于分块大小固定, 如果分块过小, 会带来更多开销; 如果分块过大, 每个计算设备最后任务完成时间可能差异较大导致某个设备任务完成时间远晚于其他设备.

比较性能表现较好的几个调度算法, 本文提出的快速分块调度算法不仅性能表现顶尖, 而且即使改变可变

参数, 性能变化不大, 更不会出现明显下降. 说明快速分块调度算法不仅具有提升推理速度的优势, 还有很好的鲁棒性, 不容易因为缺乏经验对可变参数的设定不准而导致推理性能明显下降. 根据算法原理, 可以推测出, 由于快速分块调度算法前期分块较大后期分块较小, 相比 HAT 算法更不容易造成每个计算设备最后任务完成时间较大差异, 同时分块不会过多, 所以性能和 HAT 算法一样较为优秀且总体表现好于 HAT 算法.

根据实验数据, 将同一算法在实验中取不同参数得到的推理速度数据求平均值作为该算法的平均推理速度, 比较各算法在两个数据集上的平均推理速度, 总结在表 4 和表 5 中. 表中数据是所在列的算法和所在行的算法平均推理的速度的比较, 例如 127.7% 对应的意思是单 GPU 推理速度比单 CPU 快 127.7%. 可以看出使用能在运行时根据运算设备性能动态分配任务的调度算法对 Caffe 深度学习推理速度的提升很大, 其中快速分块调度算法表现最好.

以 CPU 和 GPU 单核计算性能乘以相应核心数量, 作为理论最高性能. 定义某一算法的性能与理论最高性能的差别为: 差别=(理论最高性能−某种算法性能)/理论最高性能. 差别越小, 表示该算法的表现性能越好. 图 5 将实验中得到的几种在运行时分配任务的调度算法的性能数据与理论最高性能的差别做了比较. 结果表明, 快速分块调度表现最好, 与其余 4 个调度算法中表现最好的 HAT 算法相比, 快速分块调度算法与理论最高性能的差别在 MNIST 和 Cifar-10 这两个数据集上分别减小了 7.4% 和 21.0%, 表明该算法对计算设备性能的利用率更高.

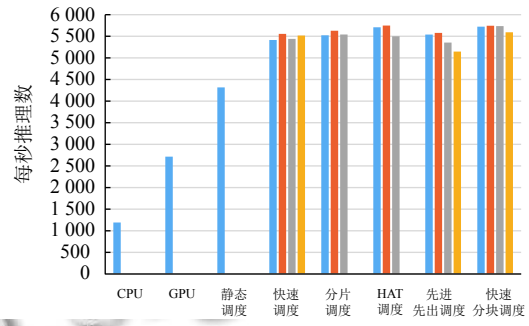

图 3 不同调度算法下 Caffe 在 MNIST 数据集上的每秒推理数

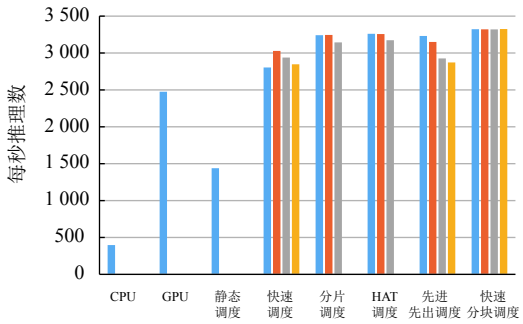

图 4 不同调度算法下 Caffe 在 Cifar-10 数据集上的每秒推理数

表 4 各种调度算法在 MNIST 数据集上推理速度表现比较 (%)

	单CPU	单GPU	静态调度	快速调度	分片调度	HAT调度	先进先出调度	快速分块调度
单CPU	—	127.7	262.3	359.9	366.8	374.2	353.4	378.1
单GPU	−56.1	—	59.1	102.0	105.0	108.2	99.1	109.9
静态调度	−72.4	−37.1	—	27.0	28.9	30.9	25.2	32.0
快速调度	−78.3	−50.5	−21.2	—	1.5	3.1	−1.4	4.0
分片调度	−78.6	−51.2	−22.4	−1.5	—	1.6	−2.9	2.4
HAT调度	−78.9	−52.0	−23.6	−3.0	−1.5	—	−4.4	0.8
先进先出调度	−77.9	−49.8	−20.1	1.4	3.0	4.6	—	5.4
快速分块调度	−79.1	−52.4	−24.2	−3.8	−2.4	−0.8	−5.2	—

6 总结和展望

为了在异构并行计算平台上提升 Caffe 框架的深度学习推理速度, 本文扩展了 Caffe 的部署程序, 使用户可以自定义编程调用 Caffe 推理. 然后将已有的异构调度算法扩展了多设备支持. 针对已有调度算法的不足, 提出了能够减少同步等待的先进先出算法、能够同时减少同步等待且减少分块次数的快速分块调度算法. 测试结果表明使用扩展的 Caffe 部署程序, 已有的快速调度、分片调度、HAT 调度等算法, 以及本文提出的先进先出调度、快速分块调度算法都能大幅提高推理时异构并行计算硬件的利用率. 其中快速分块调度表现优秀稳定, 在 MNIST 和 Cifar-10 数据集上推理速度相比 CPU 单核分别提升了 378% 和 736%, 相比单 GPU 分别提升了 110% 和 34%; 相比已有最好的 HAT 调度算法, 在 MNIST 和 Cifar-10 这两个数据集上分别减小了 7.4% 和 21.0%

的计算性能浪费.

在将来的研究中, 可以探索深度学习模型中使用的神经网络类型或者输入数据规模如何造成 CPU 和 GPU 的推理速度差异. 根据神经网络类型、输入数据规模等信息更快地选择出合适的调度算法参数来优化推理速度的提升.

表 5 各种调度算法在 Cifar-10 数据集上推理速度表现比较 (%)

	单CPU	单GPU	静态调度	快速调度	分片调度	HAT调度	先进先出调度	快速分块调度
单CPU	—	523.20	262.10	631.20	708.20	713.30	666.60	736.40
单GPU	−84.00	—	−41.90	17.30	29.70	30.50	23.00	34.20
静态调度	−72.40	72.10	—	101.90	123.20	124.60	111.70	131.00
快速调度	−86.30	−14.80	−50.50	—	10.50	11.20	4.80	14.40
分片调度	−87.60	−22.90	−55.20	−9.50	—	0.60	−5.10	3.50
HAT调度	−87.70	−23.40	−55.50	−10.10	−0.60	—	−5.70	2.80
先进先出调度	−87.00	−18.70	−52.80	−4.60	5.40	6.10	—	9.10
快速分块调度	−88.00	−25.50	−56.70	−12.60	−3.40	−2.80	−8.40	—

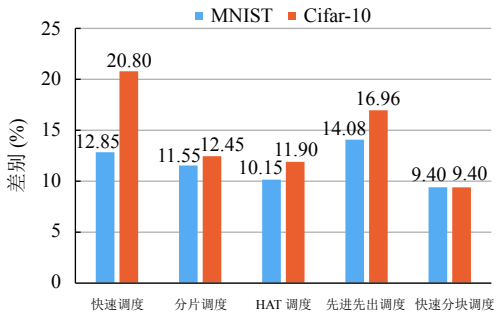

图 5 不同调度算法性能相比理论最高性能的差别百分比

参考文献

1 Bhatnagar M, Singh PK. Choice of efficient image classification technique using limited device. International Journal of Electronics and Computer Science Engineering, 2012, 1(3): 1070–1079.

2 Lonn S, Radeva P, Dimiccoli M. Smartphone picture organization: A hierarchical approach. Computer Vision and Image Understanding, 2019, 187: 102789. [doi: 10.1016/j.cviu.2019.07.009]

3 Jia YQ, Shelhamer E, Donahue J, *et al*. Caffe: Convolutional architecture for fast feature embedding. Proceedings of the 22nd ACM International Conference on Multimedia. Orlando: ACM, 2014. 675–678. [doi: 10.1145/2647868.2654889]

4 Kovalev V, Kalinovsky A, Kovalev S. Deep learning with theano, torch, caffe, tensorflow, and deeplearning4J: Which one is the best in speed and accuracy? Proceedings of the XIII International Conference on Pattern Recognition and Information. Minsk: Belarus State University, 2016. 99–103.

5 BVLC. Interfaces. https://caffe.berkeleyvision.org/tutorial/interfaces.html. (2018-08-14)[2021-03-24].

6 Roy P, Song SL, Krishnamoorthy S, *et al*. NUMA-caffe: NUMA-aware deep learning neural networks. ACM Transactions on Architecture and Code Optimization, 2018, 15(2): 1–26. [doi: 10.1145/3199605]

7 NVIDIA. Gpu-rest-engine. https://github.com/NVIDIA/gpu-rest-engine. (2018-07-21)[2021-03-01].

8 AMD. OpenCL-caffe. https://github.com/amd/OpenCL-caffe. (2018-08-31)[2021-03-01].

9 Tschopp F, Martel JNP, Turaga SC, *et al*. Efficient convolutional neural networks for pixelwise classification on heterogeneous hardware systems. Proceedings of the 13th International Symposium on Biomedical Imaging. Prague: IEEE, 2016. 1225–1228. [doi: 10.1109/ISBI.2016.7493487]

10 朱传家, 刘鑫, 方佳瑞. 基于“神威太湖之光”的 Caffe 分布式扩展研究. 计算机应用与软件, 2020, 37(1): 15–20.

11 BVLC. Using a trained network: Deploy. https://github.com/BVLC/caffe/wiki/Using-a-Trained-Network:-Deploy. (2017-03-19)[2021-02-25].

12 Wang ZN, Zheng L, Chen Q, *et al*. CPU + GPU scheduling with asymptotic profiling. Parallel Computing, 2014, 40(2): 107–115. [doi: 10.1016/j.parco.2013.11.003]

13 Chen Q, Guo MY. Task scheduling for multi-core and parallel architectures. Singapore: Springer Nature, 2017. [doi: 10.1007/978-981-10-6238-4]

14 Docker Inc. Runtime options with Memory, CPUs, and GPUs. https://docs.docker.com/config/containers/resource_constraints/. (2021-02-27)[2021-03-01].

15 Yann L, Corinna C, Christopher JCB. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/. [2021-03-01].

16 Krizhevsky A. Learning multiple layers of features from tiny images. Toronto: University of Toronto [Master's Thesis], 2009.